NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OBJECT-ORIENTED METHOD FOR INTEROPERABILITY

by

Stephen F. Shedd

September 2002

Thesis Advisor: Man-Tak Shing
Second Reader: Paul Young

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Semantic and Syntactic Object Correlation in the Object-Oriented Method for Interoperability			**5. FUNDING NUMBERS**
6. AUTHOR(S) Stephen Fletcher Shedd			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Interoperability is not a luxury, it is a necessity. Unfortunately, differences in data representation between various systems greatly complicates the task of achieving interoperability. Young's Object-Oriented Method for Interoperability (OOMI) describes a model-based, computer-aided methodology for resolving modeling differences among heterogeneous systems in order to enable system interoperability. The OOMI architecture and tool suite provide a high level of automation that will reduce the labor and complexity of integrating heterogeneous systems into a cooperative system of systems (federation of systems). The Component Model Correlation process in the OOMI architecture describes a methodology to correlate a component system's model of a real-world entity to the federation model of the same real-world entity. Once a correspondence is established, the OOMI tool suite facilitates the construction of wrapper-based translations between the component model and the federation model. These translations are then used in a run-time translator to enable interoperation between the federation of systems. This thesis describes the Component Model Correlation methodology and presents a prototype Component Model Correlator that assists an Interoperability Engineer in determining component model correspondence.

14. SUBJECT TERMS Interoperability, Component Model Correlation, Semantic Correlation, Syntactic Correlation, Heterogeneous Software Systems, XML, Neural Networks, Java			**15. NUMBER OF PAGES** 259
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OBJECT-ORIENTED METHOD FOR INTEROPERABILITY

Stephen F. Shedd
Lieutenant, United States Navy
B.S., United States Naval Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author: Stephen F. Shedd

Approved by: Man-Tak Shing
Thesis Advisor

Paul Young
Second Reader

Chris Eagle
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In today's military interoperability is not a luxury, it is a necessity. Unfortunately, differences in data representation between various systems greatly complicate the task of achieving interoperability between them. Young's Object-Oriented Method for Interoperability (OOMI) describes a model-based, computer-aided methodology for resolving modeling differences among heterogeneous systems in order to enable system interoperability. The OOMI architecture and tool suite provide a high level of automation that will reduce the labor and complexity of integrating heterogeneous systems into a cooperative system of systems (federation of systems). The Component Model Correlation process in the OOMI architecture describes a methodology to correlate a component systems model of a real-world entity to the federation model of the same real-world entity. Once a correspondence is established, the OOMI tool suite facilitates the construction of wrapper-based translations between the component model and the federation model. These translations are then used in a run-time translator to enable interoperation between the federation of systems. This thesis describes the Component Model Correlation methodology and presents a prototype Component Model Correlator that assists an Interoperability Engineer in determining component model correspondence.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

CASI	Computer Aided System Integration tool
CCR	Component Class Representation
DOM	Document Object Model
FCR	Federation Class Representation
FE	Federation Entity
FEV	Federation Entity View
FIOM	Federation Interoperability Object Model
GUI	Graphical User Interface
IE	Interoperability Engineer
IDE	Integrated Development Environment
JDOM	Java DOM
OOAD	Object-Oriented Analysis and Design
OOMI	Object Oriented Method for Interoperability
RWE	Real-World Entity
SAX	Simple API for XML
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to first and foremost thank my wife, Lauren, and son, Charlie, for supporting me through this entire graduate education endeavor. I could not have done it without their support, especially during the last few months when every step seemed to be uphill through ten feet of snow and I thought I would never finish.

I would also like to extend my thanks to my advisor, Professor Man-Tak Shing, and my second reader, Captain Paul Young, for their guidance and support in reviewing my work and completing the project. Also, thank you to CAPT Young, whose dissertation work provided the foundation for this thesis. Lastly, I would like to thank Lieutenant George Lawler for his invaluable assistance in integrating the Component Model Correlator module into the prototype OOMI IDE. Without his help, the prototype simply would not have worked.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to develop and analyze a prototype implementation of the Component Model Correlation module of the Object-Oriented Method for Interoperability Integrated Development Environment (OOMI IDE) proposed by Young [You02]. This thesis consists of five chapters.

Chapter I provides an introduction of the thesis and an overview of the Component Model Correlator.

Chapter II provides background discussions on interoperability and its inhibitors, integration of systems and theoretical foundations of object correlation in order to form a federation ontology. A detailed review of the OOMI methodology and tool suite is also presented

Chapter III discusses the component model correlation methodology within the OOMI IDE including semantic and syntactic component generation and semantic and syntactic correlation.

Chapter IV presents an implementation of a prototype Component Model Correlator module for the OOMI IDE. The software module implements the methodology discussed in Chapter III.

Chapter V provides conclusions and recommendation for future research.

B. CORRELATOR OVERVIEW

The Object-Oriented Method for Interoperability (OOMI) describes a model-based, computer-aided methodology for resolving modeling differences among heterogeneous systems in order to enable system interoperability [You02]. It consists of three basic parts. The first component is the Federated Interoperability Object Model (FIOM). The FIOM is a model used to capture the real-world entities whose state and behavior are shared between a federation of heterogeneous systems [You02]. The second component of the OOMI methodology is the OOMI IDE, which enables computer-aided,

pre-runtime construction of an FIOM. The last component is the run-time OOMI Translator. The OOMI Translator is a middleware component that uses software wrappers to resolve heterogeneities among systems, thus enabling interoperation [Lee02]. These three components of the OOMI are discussed in detail in II.D.

Object correlation is the process of identifying correspondences among different system models of a real-world entity. This is done by comparing a component model to the "standard" federation model of an entity. Once this correspondence is established, a translation can be defined resolving differences among systems. Correlation software assists the interoperability engineer in constructing and maintaining the FIOM by associating component system data models with the federated representation of real-world entities. In order to build a translation between a component system and the FIOM, we must first identify matching entities that can be translated. An English/Spanish bilingual dictionary can be used as an analogy for this process. The dictionary itself can be considered an FIOM. When a person wishes to translate the English word "son" into Spanish, they look up the word in the English portion and find the Spanish equivalent, in this case "hijo." The act of looking up the translation and converting the English word into Spanish is similar to the function of the run-time OOMI Translator. But first, the dictionary editor must identify the associations between English and Spanish words, create the translations and then add the translations to the dictionary. The act of finding these associations describes the process of correlation; someone in the process must discover that the Spanish word "hijo" is equivalent to the English word "son."

In a federation of heterogeneous systems, finding associations between component system and federation models in order to build translations is far more complex. The OOMI IDE attempts to automate this process and the correlation software is a crucial module for automating the construction and maintenance of a FIOM. During construction of a new FIOM, an interoperability engineer can create Federation Class Representations (FCRs) that provide a federation model of a real-world entity from schemas that define real-world entities. When a new system is added to the federation, the interoperability engineer uses a Component Class Representation (CCR) that provides a component model of the real-world entity to search the FIOM for a matching FCR. The search is performed

in two steps. First, a keyword search associates entities based on semantics, or meaning. Next, neural networks are used to filter keyword associations by syntax, or structure of an entity. The goal of correlation is to return a small set of FIOM FCRs that closely resemble the component system's CCR. If there is a match, translations are constructed that translate between the component system model of an entity and the federated view of the entity. If there is no match, the interoperability engineer can add a new federated entity to the FIOM and complete the construction of the translations.

In order to comprehend the search mechanisms of the correlator software, it is important to understand the structure of classes in object-oriented programming. Classes contain attributes and operations. Each attribute and operation has descriptions and certain structural characteristics that can be used in the correlator software. For example, keywords can be captured from attribute names and description fields contained in the schema or interface definition. Syntactic or structural information can be derived from characteristics such as data type and frequency of occurrence.

An important feature of the OOMI IDE and Correlator software is the use of XML schemas to represent a CCR and FCR. An XML schema can easily represent a component systems interface and facilitate computer automation. Further, an XML schema can be used to represent an object-oriented class construct in terms of the data contained in the class and the structure of the class itself. Complex data types can be decomposed into simple types and characteristics such as enumeration values can be defined. Thus, an object-oriented FIOM can be constructed that uses an object to model all entities.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INTEROPERABILITY AND ITS INHIBITORS

In today's military, interoperability is not a luxury…it is a necessity. According to the 2001 Quadrennial Defense Review:

> U.S. forces require the ability to communicate not only with one another, but also with other government agencies and allies and friends. Such joint and combined interoperability requires forces that can immediately "plug" into the joint battlefield operating systems (command and control, intelligence, fire support, logistics, etc.) and perform effectively. These forces need compatible systems with interoperable standards, doctrine, tactics, techniques, and procedures. [QDR01]

System interoperability involves not only the ability of systems to exchange information but also includes the capability for interaction and joint execution of tasks. [LISI98, Pit97]. Achieving this level of interoperability is difficult for two major reasons. First, legacy systems exist throughout the entire Department of Defense establishment that were designed without including provisions for interoperation. Like it or not, their existence is often necessary and in many cases it is prohibitively expensive to upgrade or retrofit legacy systems to make them capable of interoperation. The second difficulty in achieving interoperability is rooted in the fact that components were developed independently, with little or no requirement for interaction at a federation level. For example, components may have had requirements for intra-service interoperability but no requirement for inter-service interoperability. As a result, component systems that are being tied together may have different architectures, different hardware platforms, different operating systems, different host languages and different data models. These differences make the task of integrating systems time consuming and complex.

1. Legacy Systems

Throughout the Department of Defense, numerous legacy systems continue to perform various important functions, operating in a stand-alone configuration or within a closed system. As outline by [Pug01], there are many reasons why these non-integrated, legacy systems exist:

- The original intended system use was so restricted in scope that there was no need to integrate. Unfortunately, this paradigm was widespread in all the services and resulted in numerous "stove piped" systems. After all, why would an Air Force C4I system need to interoperate with a Navy C4I system?
- The original system was classified at a security level above other systems, which precluded interconnection with systems at a lower security classification.
- Personnel in the acquisition and development process, both contractors and government employees, failed to consider future integration for the system.
- The original system designers never imagined that their system's lifecycle would extend as long as it has. Subsequently, many unforeseen uses of the system and the benefits of connecting with new or previously unknown systems were not provided for in the original design.

Not surprisingly, many of these legacy systems run on proprietary hardware and execute very old, proprietary software. The utopia of interoperability would be an environment where these different systems could communicate and share their information, a "system of systems." Clearly, seamless integration of our independent systems would greatly increase our war fighting capability.

2. Heterogeneity and Modeling Differences

Young proposes a clear distinction between a federation of systems and an integrated system of components. The term "integrated system of components" is used to describe an interconnected compilation of homogenous components produced by a development team that shares common objectives and has a common view of the problem environment being modeled [You02]. In the personal computer domain, the Microsoft Office® software suite can be used as an example. Each application in the Office® suite is a homogeneous component but developed under the common "office" view. Thus integration is achieved.

On the other hand, the term "federation of systems," or "system federation," describes an interconnected collection of independently developed heterogeneous systems or components [You02]. Most of the systems in the Department of Defense fall under this classification. Each service develops a number of heterogeneous systems and the entire goal of joint warfare is to interconnect these systems and overwhelm the enemy.

Young [You02] describes eight types of heterogeneity that can be used to characterize potential differences in independently developed systems:

- Heterogeneity of Hardware and Operating Systems – Differences in the hardware and operating system platforms encountered when integrating autonomously developed systems can often lead to differences in the physical representation of information. For example, an integer may be represented in 16 bits on system A but 32 bits on system B. Also, central processing units can implement arithmetic in different ways, which can lead to further differences in the representation of information.

- Heterogeneity of Organizational Models – Heterogeneity of organizational models refers to differences in the conceptual models used by autonomously developed systems. A common difference is the model used for analysis and design such as the use of Object-Oriented Analysis and Design (OOAD) versus Structured Analysis and Design (SAD). In the database domain, this type of heterogeneity can result from the use of relational, hierarchical or object structured databases.

- Heterogeneity of Structure – Structural heterogeneity refers to variations in the structure of how information is arranged among systems using the same organizational model, including differences in structural composition, possible schema mismatches, and variations due to the presence of implied information. For example, a ship can be modeled as a complex data type in one system and in another, the same entity is modeled as a string containing a contact track number.

- Heterogeneity of Presentation – Domain mismatch problems, the use of different units of measure, differences in precision, disparate data types, and different field lengths or variations in integrity constraints are types of heterogeneity of presentation. Two common examples of this type of heterogeneity are difference in position representation and differences in altitude representation. One system may use latitude and longitude for position

and height above the ellipsoid (HAE) for altitude while another uses the military grid reference system (MGRS) for position and mean sea level for altitude.

- Heterogeneity of Meaning – Heterogeneity of meaning refers to imprecise natural language characterizations of real-world entities. Examples include homonyms, synonyms, and abbreviations. This type of heterogeneity is a chronic problem in the Department of Defense. One acronym can have multiple meanings depending on the domain in which it is used.

- Heterogeneity of Scope – Heterogeneity of scope describes the differences in the information used to model a real-world entity that arise from disjoint perspectives on the attributes a given application needs to capture about a real-world entity. For example, the development team of a Marine Corps Targeting System may model an enemy tank with the attributes *type*, *range* and *rateOfClosure*. The developers of an Intelligence system may model the same enemy tank with the attributes *classification, parentUnit*, and *fuelCapacity.*

- Heterogeneity of Level of Abstraction – Heterogeneity of level of abstraction results from differences in the level and degree of aggregation of atomic data elements. For example, a training application used by a numbered fleet may model a battle group as a single unit. The fleet staff is focused on the training of the battle group in the aggregate. On the other hand, a similar training application used by a battle group staff models the battle group as a collection of distinct units.

- Heterogeneity of Temporal Validity – Heterogeneity of temporal validity refers to differences in the time used by two models to observe or record the state of a real-world entity. A difference in the length of time data remains valid is another source of heterogeneity of temporal validity. For example, a Battle Group command and control system may require a second-by-second update on the positions of unit tracks while a Fleet level command and control system only needs a minute-by-minute or even hour-by-hour update.

3. Differences in View

The eight classes of heterogeneity are further categorized by Young into differences in view and differences in representation.

- Differences in view – Different systems can model different characteristics of a real world entity (*heterogeneities of scope, level of abstraction* and *temporal validity* [Wie93,Young02]). For example, in Figure II-1, three different views of a ground combat vehicle are shown. Each system modeling the ground combat vehicle does so in a manner that best fits the requirements of that system. In view 1, systems A and B capture information about the entity's *type*, *position*, *time* and *range.* In contrast, system D views the ground combat vehicle in terms of *type, position, time and status*. Lastly, system C provides a third view by modeling the entity's *type*, *position*, and *time.* Although the three views of the ground combat vehicle are similar, they are distinct in scope and level of abstraction and temporal validity. [Lee02]
- Differences in representation – In addition, the same characteristics may be represented differently (*heterogeneities of organizational models, structure* and *presentation* [Wie93, Young02]). For instance, as shown in Figure II-1, system A represents the attributes *position* and *time* in latitude/longitude coordinates and Greenwich Mean Time (GMT) respectively. In contrast, system B represents *position* in military grid reference system (MGRS) coordinates and *time* in Local Mean Time (LMT).

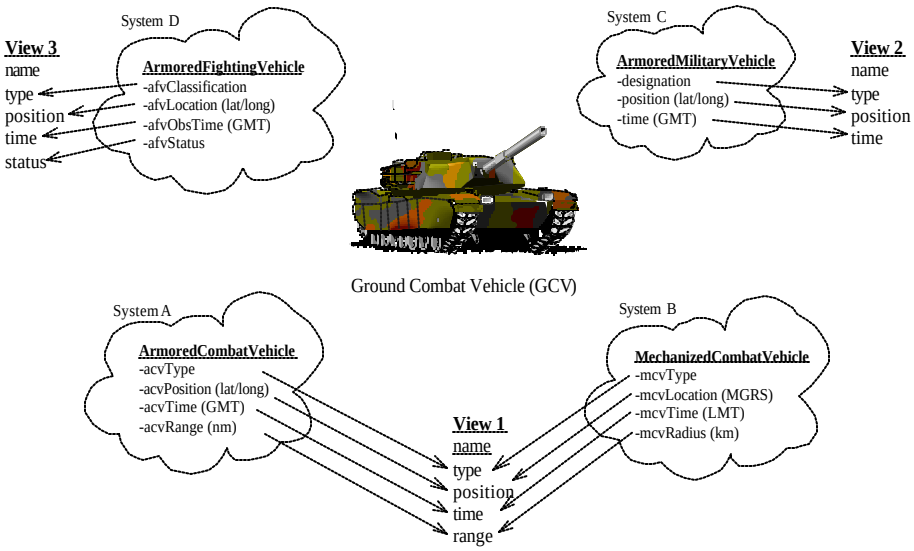


Figure II-1. Differing views of a real-world entity. [from You02]

B. INTEGRATING SYSTEMS

There are three basic ways to integrate heterogeneous systems to account for differing views: change one or both system's hardware, change one or both system's software, and through organizational change. [Pug01] Organizational change seems to be the predominant method although it is the least desirable. Organizational change is epitomized in the euphemism, "rather than develop a system we need, here is a system, now figure out where we need it." The manifestations of organizational change are work-arounds and *ad hoc* methods to exchange useful, or even required information between systems. This results in a degradation of standards and increases the complexity of dealing with integration. For mission-critical systems, work-arounds and ad hoc methods are a very serious safety issue since they could potentially cause a system to behave in unintended and untested ways.

Hardware changes are sometimes the only way to achieve system integration and this method can be relatively inexpensive. However, hardware cannot be used to resolve heterogeneities of modeling differences and does not contribute to the concept of interoperability. Software changes seem to be the best approach for achieving system integration and achieve interoperability. Software is inherently "lightweight" by its very

nature. However, poor software engineering practices and development methods can present challenges in using software for integration.

The foundation of software integration lies in external interfaces. A system's external interface is a contract of services between that system and the other systems or components that may need those services. External interfaces are almost always well defined and well documented and usually include the type of the provided data, accepted values, and formatting information. Interfaces encapsulate the implementation of features and allow a system to evolve. As long as the external interfaces do not change, the system is virtually free to change in any way, shape, or form.

There are three approaches to system integration through software. In the first approach, integrators design and develop a system for integration and interoperability from the outset of the project. This requires knowledge of all the external interfaces for all the systems for which interoperability is required. The second approach is to reengineer legacy code to a newly identified external interface. The third approach is to create middleware in the form of software wrappers, which is essentially a translator between two external interfaces. There are many benefits to this approach that will be discussed shortly.

In the first approach for integration with software, integrators design and develop a system for integration and interoperability from the outset of the project. When software is initially developed, the requirements should specify the level of interoperability required by the system under development. The requirements should also list the specific systems that the system under development will need to interoperate with. The system designers then obtain the external system interfaces of those systems and design the new system to conform to those external interfaces. As long as the developers of the new systems adhere to the external interfaces of the other systems, the new system should be able to integrate and interoperate smoothly. However, there are a few, rather large problems with this approach. First and foremost, whoever develops the requirements must know in advance all the systems that are likely candidates for interoperation or integration with the new system. As stated in the discussion of legacy systems, it is not possible to predict all the future uses of a system. Even with the star government-contractor team, not

all uses will be identified. When adding a new system that does not include provisions for interoperation with the existing system, reengineering will be required. This does not mean that interoperability should not be considered from the outset of a project. On the contrary, it is essential. However, the reality is that a new system will eventually be asked to perform in unintended ways, which leads to the last two approaches to software integration.

Reengineering legacy code is problematic. In this method, programmers modify the source code of a system to meet the specifications of another external interface. There are many problems involved. To start, the source code for many COTS and legacy systems is simply not available. Secondly, the time and effort expended in comprehending and changing old code, often written in arcane programming languages like CSM-2, often greatly exceeds the time and effort that would be required to rewrite the code from scratch. Lastly, the resources needed for regression testing of the legacy code can be prohibitive. Despite these issues, there may be times when modifying legacy code is the only option. Fortunately, software wrappers usually provide a better alternative.

Software wrappers are middleware that translate one external interface to another. These small pieces of software offer a lightweight solution to converting modeling differences between systems. In a simple case, software wrappers may only provide a mapping of data elements between two systems with no data conversions. In most cases, however, software wrappers provide data conversion routines to ensure data elements from systems conform to the external interface of another. In this fashion, wrappers are very useful. The major drawback in using software wrappers is the scalability for large-scale systems. Whenever a system is added to a domain of federated systems, a programmer has to develop a new wrapper for each system already attached. Figure II-2 illustrates this point. Federation A and B consist of N component systems. Each components system requires a wrapper to the other systems, resulting in $N(N-1)$ or N^2-N wrappers. While Federation A has a total of 6 wrappers, Federation B needs 12 wrappers. This quadratic explosion of wrappers makes pair-wise wrapper construction for integration a poor choice.

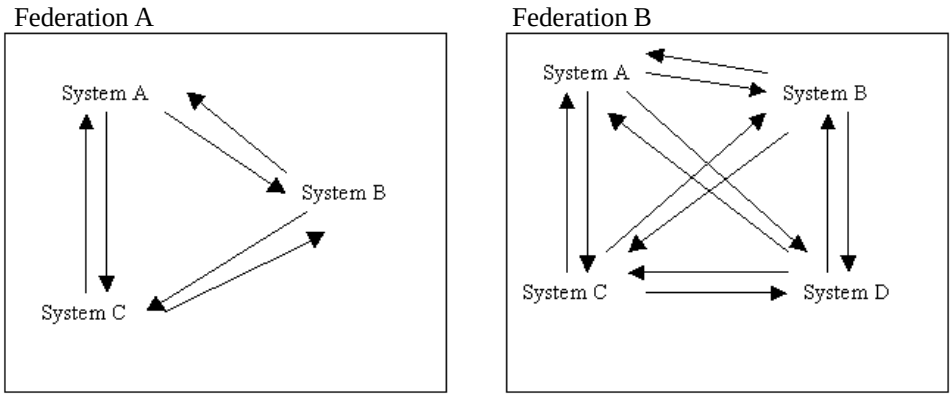


Figure II-2. Pair-wise software wrapper construction. [from Pug01]

Alternatively, we can avoid combinatorial wrapper explosion by modeling the federation of systems in a star-like topology. In this case, every node that wants to communicate with another node first translates the communication to a common representation of the communication or real-world entity. This common representation can be thought of as the *"lingua franca"* [Pug01] for the federation and may be a single system's representation of entities or may be a domain representation of entities. With this topology, each component system needs only two software wrappers, one to translate from its own system specific representation to the common representation, and one to translate from the common representation to the system specific representation. In comparison to the federation shown in Figure II-2, the number of software wrappers drops to 2N instead of N(N-1). Figure II-3 shows the benefits of using a common representation for software wrappers in a federation of systems.

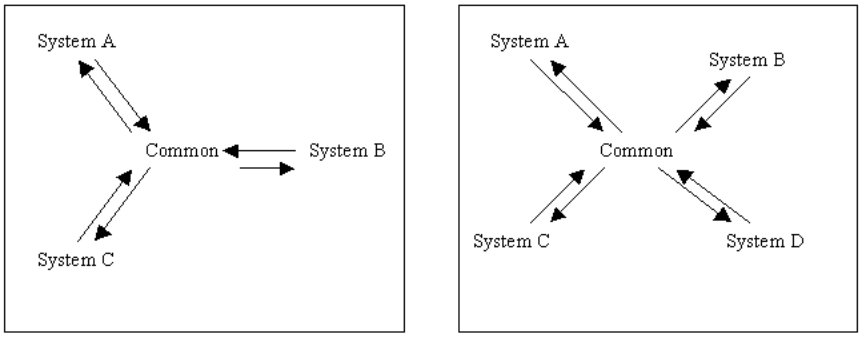


Figure II-3. Software wrappers for a common representation in a federation of systems. [from Pug01]

C. OBJECT CORRELATION IN A FEDERATED ONTOLOGY

A federation of heterogeneous systems based upon a common representation or model, as shown in Figure II-3, is an ideal approach to integrate systems and promote interoperability. Information can then be exchanged using the common representation, with each system only concerned with the translation between the common model and its own internal model. New component systems can be added to the federation by creating a set of wrapper-based translations between the component system's model of the information and the common model. As component systems evolve, translations can be modified, added or deleted.

In order for systems in a federation to share information, system designers must resolve differences in what information is being shared, described as heterogeneity of scope, level of abstraction, or temporal validity, and differences in how information is represented, described as heterogeneity in structure, presentation, meaning, hardware/operating system, and organizational models. One method that can be used to model a common representation of information between federated systems is the use of an ontology. An ontology is "a knowledge base consisting of entities and relationships with abstraction, inference and typing mechanisms" [HL96, p.8]. As stated by Young:

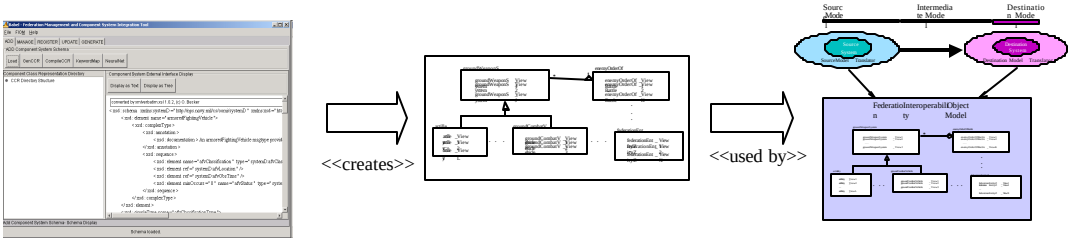
> The federated ontology is a global schema to which attributes and operations in a component system are associated via syntactic and semantic information. These associations between the global schema and component system enable translations between the component and global schema model of an attribute or operation. [You02]

The main benefit of using an ontology is its ability to resolve most types of heterogeneity. The primary drawback to this method is the time and effort needed to construct the required ontology. A federation of heterogeneous component systems brings together many models of real-world entities that results in many views as discussed in Section II.A.3. Correlation between these disparate views of real-world entities is required whether the process occurs manually or through computer automation. The correlation is necessary in order to identify entities that can be translated to and from one another. Manual correlation between different models of each real-world entity can be

difficult, time-consuming and cost-prohibitive. The process of integrating databases involves the same correlation issues and offers a good example of the resources required for manual correlation. A group from GTE began a database integration process involving 27,000 data elements from 40 applications. On average, the members of the group required 4 hours per element to extract, document and identify matching elements from the other applications. [LC02] Given large, complex systems, the task of manually comparing data elements and attributes is unreasonably large. Clearly, in order to create and maintain a large-scale federation of systems based upon wrapper-based translations, a computer-automated correlator is required.

D. OBJECT-ORIENTED METHOD FOR INTEROPERABILIY (OOMI)

The Object-Oriented Method for Interoperability (OOMI) [You02] describes a model-based, computer-aided methodology for resolving modeling differences among heterogeneous systems in order to enable system integration and achieve interoperability without the need to rework legacy code. The OOMI consists of three elements as illustrated in Figure II-4.



OOMI Integrated Development Environment (OOMI IDE)

Federation Interoperability Object Model (FIOM)

OOMI Translator

Figure II-4. Components of the OOMI. [from You02]

The first element is a general model of the interoperation between systems called a Federation Interoperability Object Model (FIOM). The second element is the OOMI Integrated Development Environment, a Computer Aided System Integration (CASI) tool that provides computer automation for the construction of an FIOM, data element and entity correlation, and construction of translations between the federated ontology and the

component systems. The last element is the OOMI Translator, which performs run-time translations between components systems.

1. FIOM

The Federation Interoperability Object Model (FIOM) forms a common ontology of Real-World Entities (RWEs) that are shared among a federation of systems. The relationship of the FIOM to the other major components of the OOMI methodology is shown in Figure II-5.

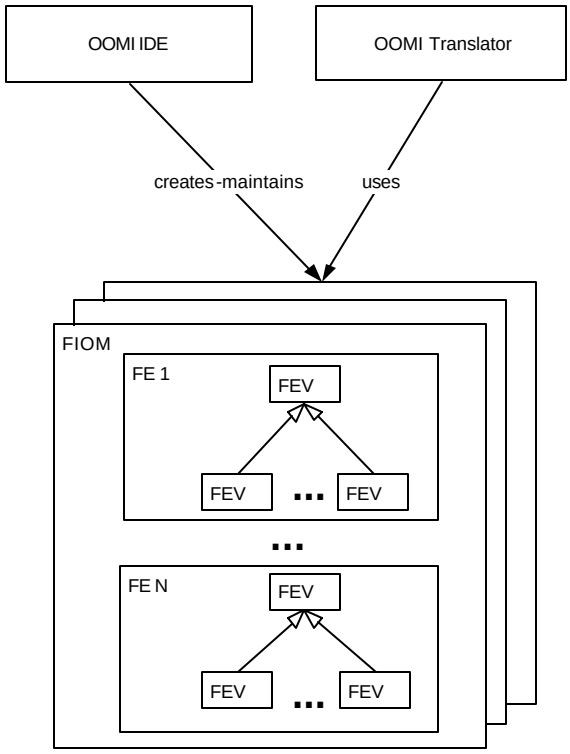


Figure II-5. Relationship of the FIOM with the major OOMI components. (after [Lee02])

The FIOM is composed of one or more Federation Entities (FEs), each providing an abstract representation of a real world entity (RWE). The FE comprises a hierarchical tree of one or more Federation Entity Views (FEVs) forming an inheritance hierarchy of FEVs within the FE. Each FEV represents the view that a component system or group of component systems has of the RWE.

Each FEV contains exactly one Federation Class Representation (FCR) that models the standard representation of the view. The FCR serves as the intermediate representation for translation between the source and destination systems. An FEV also contains one or more Component Class Representations (CCRs). The CCR is a direct mapping of a component system's model of the RWE. Translations between the FCR and CCR are defined within the FIOM. Because an FEV contains exactly one FCR, translations are only need to be created between each component system and the FCR. Figure II-6 illustrates the components of an FEV.

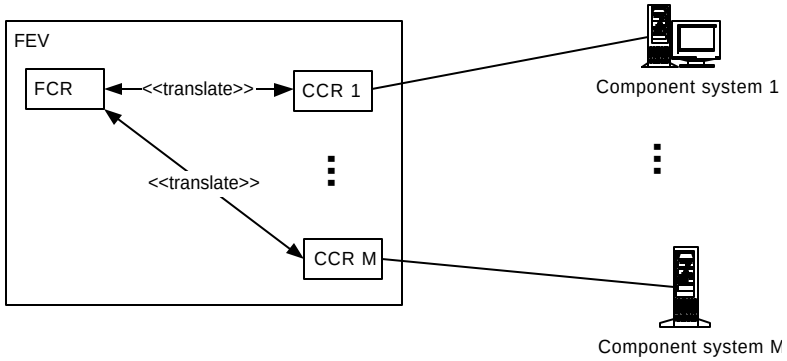


Figure II-6. FEV, FCR, CCR and Component Systems. [from Lee02]

An FCR and CCR are modeled as an object-oriented class construct as depicted in Figure II-7. Each class contains the attributes and operations that represent the state and behavior of the RWE. The FCR describes the RWE from the federated point of view and the CCR describes the RWE from the component system point of view.

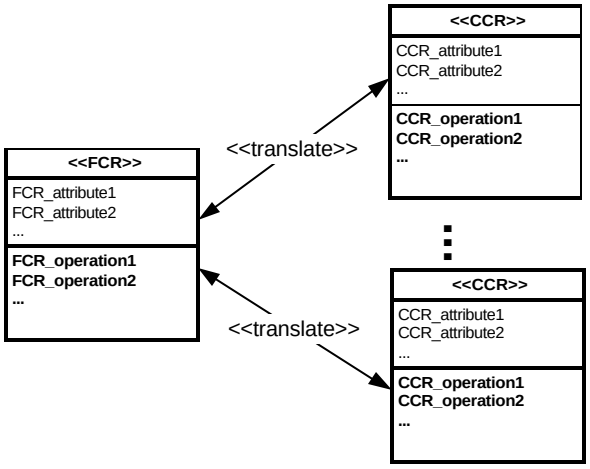


Figure II-7. FCR and CCR constructs. [after Lee02]

An FE representation of the ground combat vehicle described in Section II.A.3 is depicted in Figure II-8. Each of the three views depicted by the systems becomes a separate FEV for the groundCombatVehicle FE. Notice, however, that each of the four component systems has a corresponding CCR with the FE hierarchy. System A and B, although represented by distinct CCRs, both view the groundCombatVehicle in the same manner as the gCV_View1_FCR.

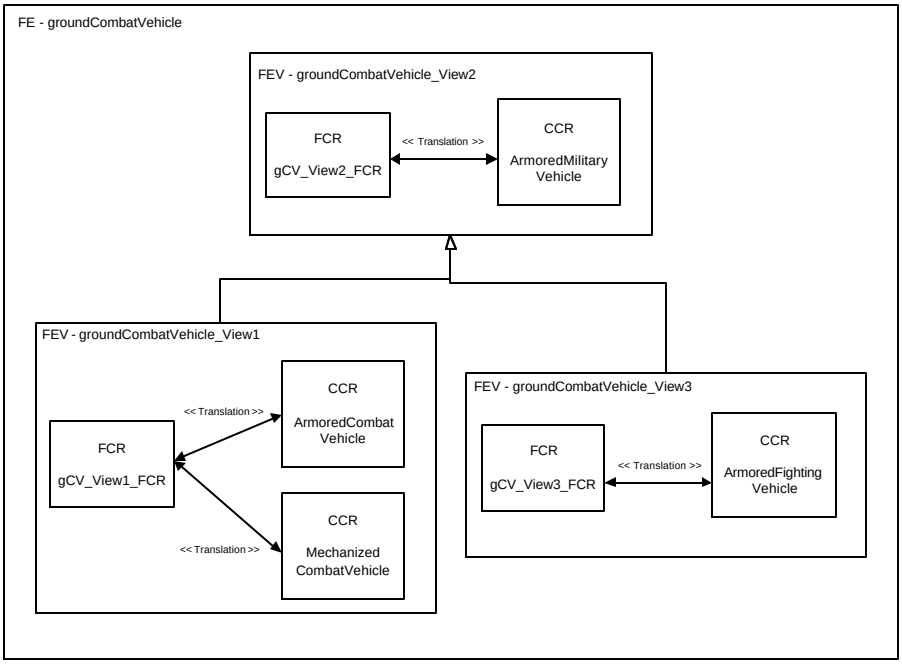


Figure II-8. FE representation of the ground combat vehicle.

2. OOMI IDE

The OOMI Integrated Development Environment is a graphical user interface (GUI) based, Computer Aided System Integration (CASI) tool used by the Interoperability Engineer. The OOMI IDE provides computer automation in order to:

- Administer multiple FIOMs
- Specify different views of a real-world entity resulting from the different perspectives each component system has of that entity
- Construct and maintain an inheritance hierarchy relating the different views of a real-world entity
- Define standard federation representations of the real-world entity views (FEV and its defining FCR) and the various component system representations (CCR)
- Integrate new components systems by finding correlations between component system representations and federation representations in the FIOM.
- Create the translations between a CCR and an FCR.

3. OOMI Translator

The OOMI Translator performs the run-time translations between component systems and the federated ontology. The FIOM, constructed for a specified federation of component systems during the pre-runtime phase, is used by the OOMI Translator at run-time to reconcile differences in real-world entity view and component system representation. The Translator receives source data or messages from a component system and retrieves the correct translation to convert the information to the federated representation. The translator than identifies the destination component system and retrieves the correct translation to convert the federated representation to the model recognizes by the destination system. See [Lee02] for an in depth discussion of the prototype OOMI Translator.

4. FIOM Framework

The FIOM Framework developed by Lee [Lee02] is a set of Java interfaces and partially implemented Java classes used for the implementation of the OOMI IDE. This section provides a summary of the FIOM Framework with specific focus on the components that are relevant to correlation within the OOMI IDE. This background information is given in order to support the component model correlator methodology discussion of Chapter III and the implementation of the component model correlator presented in Chapter IV.

The FIOM Framework, as defined in the package *mil.navy.nps.cs.oomi.fiom*, provides the functionalities required for implementing the OOMI IDE. The set of Java interfaces in this framework defines the objects modeling the FIOM. The set of Java classes defined in the *mil.navy.nps.cs.oomi.impl* package implement these interfaces. The *OOMIDatabase* interface in the package mil.navy.nps.cs.oomi is used by the OOMI IDE to handle a collection of FIOMs.

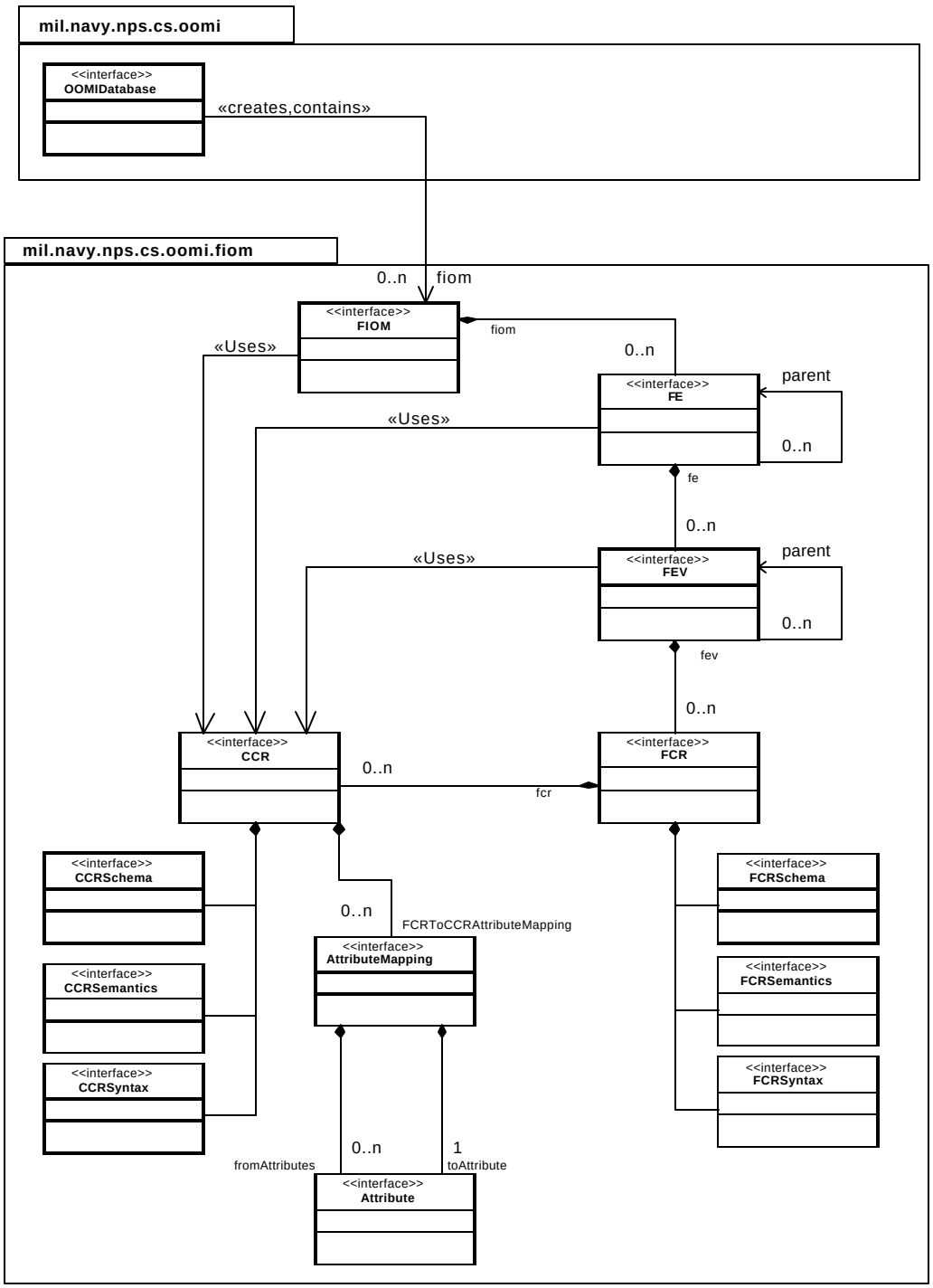


Figure II-9. mil.navy.nps.cs.oomi and mil.navy.nps.cs.oomi.fiom packages. [from Lee02]

As stated, the interfaces in the mil.navy.nps.cs.oomi.fiom package are implemented in the *mil.navy.nps.cs.oomi.impl* package. The naming convention for the

Java classes in this package is simply the name of the interface with "Impl" concatenated to the end. For example, the Java class implementing the *FCR* interface is named *FCRImpl.* Table II-1 identifies the interfaces and corresponding implementations that are relevant to the correlation process discussed in this thesis. Modifications to the FIOM framework to support the implementation of the component model correlator are outlined in Section IV.A.2.

Table II-1. Components of the FIOM Framework relevant to the OOMI IDE correlation process.

Interface	Implementation	Description
FCR	*FCRImpl*	Provides the meta information of an FCR, composed of *FCRSchema*, *FCRSemantics* and *FCRSyntax* interfaces
CCR	*CCRImpl*	Provides the meta information of a CCR, composed of *CCRSchema*, *CCRSemantics* and *CCRSyntax* interfaces
FCRSchema	*FCRSchemaImpl*	Contains the complete information on the attributes of the FCR
FCRSemantics	*FCRSemanticsImpl*	Contains the semantic components of the FCR that are used in the semantic search phase of the component model correlation process. See Section III.E
FCRSyntax	*FCRSyntaxImpl*	Contains the syntactic components of the FCR that are used in the syntactic search phase of the component model correlation process. See Section III.F
CCRSchema	*CCRSchemaImpl*	Contains the complete information on the attributes of the CCR
CCRSemantics	*CCRSemanticsImpl*	Contains the semantic components of the CCR that are used in the semantic search phase of the component model correlation process. See Section III.E
CCRSyntax	*CCRSyntaxImpl*	Contains the syntactic components of the CCR that are used in the syntactic search phase of the component model correlation process. See Section III.F

III. COMPONENT MODEL CORRELATION IN THE OOMI IDE

A. THEORETICAL FOUNDATIONS OF CORRELATION

As discussed in Section II.C, a federation of heterogeneous component systems can include many models of real-world entities (RWE), which results in numerous, sometimes disparate views of the entity. A federated ontology can be used to model a common representation of the RWE in order to promote interoperability among the heterogeneous systems. Component model correlation is required to resolve modeling differences among systems in order to assist the interoperability engineer in building translations between the component system and the federation. This section is a summary of the detailed work by Young [You02] and Pugh [Pug01]. The basis for determining correlation effectiveness is discussed followed by the theoretical foundations of correlation and the applicability to interoperability. These topics are included as background information to support the discussion of the component model correlation methodology in the rest of Chapter III and the implementation of the component model correlator presented in Chapter IV.

1. Correlation Effectiveness

Salton and McGill [SM83] provide six evaluation criteria for determining the effectiveness of information retrieval systems: precision, recall, effort, time, presentation, and coverage. All six are important to the correlation of similar entities within the FIOM; however, precision and recall are the foremost measures of correlator effectiveness. [You02].

Precision is the ratio of relevant entities returned to the total number of entities returned. In a perfect system this number will be 1.0 signifying an exact match.

Recall is the ratio of relevant entities returned to the actual number of relevant entities in the search space. High recall ensures that you have not set your threshold for correctness so high that possible matches are discarded. However, recall should be set high enough to avoid clutter and noise in the return values.

Effort is the amount of physical and intellectual work needed to perform the correlation.

Time is the measure of how long (either in CPU epochs or real elapsed time) it takes to perform the correlation.

Presentation is the method that the query uses to present data to the user. Different types of presentation may include a ranked list, where unlikely matches are included or a single "best match" candidate.

Coverage is the measure of the search space the algorithm is able to correctly query.

2. Classical Approaches For Correlation

Classical approaches to correlation include browsing, keyword matching and multi-attribute searches. The major benefits of these approaches is the availability of information required for correlation, the relative simplicity of the various approaches, and the user's ability to understand the use of the approach. The disadvantage of the classical approaches is that they do not consider component behavior and subsequently do not achieve high values for precision and recall. Of the three approaches, keyword matching has the greatest applicability to component model correlation within OOMI [You02], [Pug01].

3. Formal Specification Approaches For Correlation

Formal specification approaches for correlation include:

- Syntax-based methodologies – Match components based on a composition and structure. Signature matching in software reuse is an examples a syntax based approach [You02].
- Semantics-based techniques – Attempt to use behavior to establish correlation between elements.
- Combination of syntactic and semantic methods – A multi-level filtering approach to correlation.

The main problems with formal specification approaches are the difficulty of writing formal specifications and the time required to conduct semantic matching. Formal specification writing involves knowledge of a formal specification language and basic knowledge of discrete mathematics. Most developers are not familiar or comfortable with formal methods. Time problems with semantic matching are a result of the computationally intensive theorem proving approaches for semantic matching techniques. As stated by Young:

> The time and effort required to augment existing systems with formal specifications describing their behavior would outweigh any savings gained from the introduction of computer aid to the solution of the correlation problem. [You02]

4. Artificial Intelligence Approaches For Correlation

Artificial Intelligence approaches for correlation include:

- Natural language techniques– full-text information retrieval to extract syntactic and semantic information needed to establish correspondence between component models.
- Neural networks techniques – attribute correspondence between components through the use of component meta-data and trained neural networks. The technique uses neural networks to learn the similarities among data from field specifications and data content.

These two artificial intelligence approaches offer the greatest potential for application to component model correlation. In particular, the use of neural networks has been successful in database integration efforts, where data matching dosely relates to component model correlation. Specifically, a tool called SEMINT successfully uses neural networks to determine semantic correspondence of data elements based on metadata and data element instances. [LC02] The process used by the SEMINT tool involves 5 steps:

1. Metadata in the form of schema information and data content statistics is extracted from an individual database using DBMS specific parsers.

2. The metadata is normalized as a series of attribute vectors containing values of data content-based discriminators for each attribute. The attribute vectors are input to a self-organizing map algorithm that categorizes attributes into cluster centers according to their proximity to other attribute vectors on the map.

3. The cluster centers are used to train a three-layer backward propagation neural network to recognize attribute categories.

4. The trained neural network is provided the attribute information from another database from which it provides the similarity between each input attribute of this new database and each attribute category from the original database used to train the network.

5. System users check and confirm the similarity results returned by the trained network. [LC00]

The backpropagation neural network described in the SEMINT process is a powerful artificial intelligence tool used for complex logical operations, pattern association and optimization problems. A backpropagation neural network, illustrated in Figure III-1, is a type of neural network that is trained to produce the same output given a specific set of input.

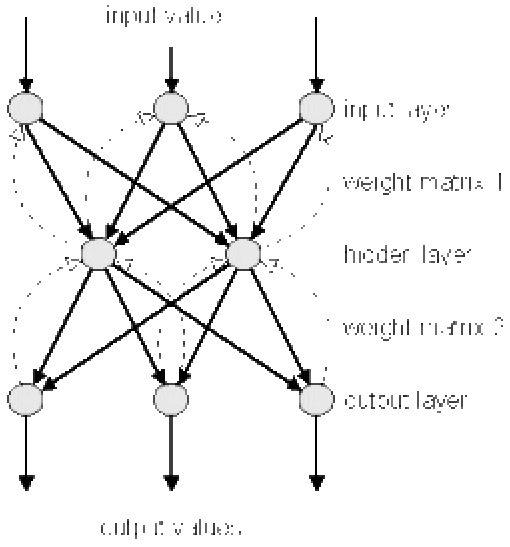


Figure III-1. Illustration of a backpropagation neural network.

In other words, the network is trained to output "y" whenever it receives "x" as input. In order to train itself, the backpropagation neural network uses a supervised learning algorithm called the backpropagation learning algorithm. As shown in Figure III-1, a neural network consists of one or more layers of nodes that emulate the action of neurons in the human brain. In a multi-layer neural network, a separate weight vector attaches a node to each successive node in the next layer. The backpropagation learning algorithm uses a computed network output error to change the weight values of the network in a backward direction. To get the network output error, a forwardpropagation phase must be done first. During forwardpropagation, the values of an input vector are passed into the nodes of the input layer of the neural network. If activated, the nodes then propagate the values to the next layer of nodes via the weighting vectors. For each node in the next layer, the input value is calculated as the sum of the weighted inputs. This process continues for every node layer in the neural network. At the completion of a single forwardpropagation pass, the nodes in the output layer provide an output vector of values, which are the results of the neural network. This output can then be compared to the output the network is trained to produce in order to perform pattern matching.

While propagating in the forward direction, the nodes are activated using a sigmoid activation function. The formula of a binary sigmoid function is

$$f(x) = \frac{1}{1+e^{-y}} \qquad \text{[Eq 3.1]}$$

where y is the input value to the particular node. This is the most typical activation function and has the range [0,1]. [Fau94] Another common activation function is the bipolar sigmoid, which has a range of [-1,1] and is defined as

$$f(x) = \frac{2}{1+e^{-y}} - 1 \qquad \text{[Eq 3.2]}$$

where y is again the value of the input to the particular node. [Fau94]

Training a neural network by backpropagation involves three stages: the forwardpropagation of an input training pattern, the backpropagation of the associated error between the desired output and actual output, and the adjustment of weights. Figure

III-1 shows the forwardpropagation as solid lines and the backpropagation as dashed lines. To begin training the net, the initial weights of weight matrix 1 and weight matrix 2 are randomized to values in the range [-0.5,0.5]. [Fau94] In the forwardpropagation, the input vector is passed into the input layer of neurons. Each neuron processes the input in accordance with the activation function and propagates the results to weight matrix1. Each output from the input layer is sent to each node of the hidden layer. In general, the number of hidden layer nodes should be

$$Num_{hidden} = \frac{Num_{output} + Num_{Input}}{2} \qquad \text{[Eq.3.3]}$$

Before the value arrives at the input of the hidden layer, it is multiplied by the value of the corresponding path of weight matrix 1. The actual input to the hidden layer is the sum of the weighted outputs from the input layer. The hidden layer nodes then use the activation function to process their inputs propagate the results to weight matrix 2. Like the hidden layer, the actual input to each output layer is the sum of the weighted outputs from the hidden layer. The output nodes then process the input using the activation function and produce an output vector. The output vector is compared to the desired output included in the training date and node errors are computed.

Using the backpropagation technique, the errors calculated between the actual output and the desired output are propagated backwards through the network. Using the generalized delta rule [Fau94], each value in weight matrix 1 and weight matrix 2 are given an error information term and a weight correction term. When all the weight correction terms have been calculated the weights are updated and the next input vector is passed through the network. This process is repeated until the network produces results that are within the error tolerance set for the output error. An epoch is one cycle through the entire set of training vectors. Typically, many epochs, ranging from hundreds to thousands, are required for training a backpropagation neural network. This makes the use of these networks computationally expensive, but often times beneficial. [Fau94]

B. OOMI IDE COMPONENT MODEL CORRELATOR METHODOLOGY

The OOMI IDE component model correlator is a module within the OOMI IDE that is responsible for establishing correspondences between component and federation models of a real-world entity in the FIOM. Figure III-2 illustrates the position of the component model correlator module in relation to the other major components of the OOMI IDE. As presented in Section II.C, an FIOM describes a federated ontology for a federation of heterogeneous component systems. When a new component system is added to a federation, the interoperability engineer must add each source system Component Class Representations (CCR) to a matching FIOM Federated Entity (FE) described by one or more Federated Entity Views (FEV). In order to assist an interoperability engineer in this task, the component model correlator is used to search the FIOM for FEVs that match a source system's CCR. If a match is found, then a translation can be constructed between a CCR and the FEV's Federated Class Representation (FCR). Recall from Section II.D.4that an FEV contains one and only one FCR, the FCR being the actual class construct used to represent the "standard" model of a real-world entity. If no match exists, the interoperability engineer can either build a new Federation Entity (FE) and Federation Entity View (FEV) or add a new FEV to an existing FE.

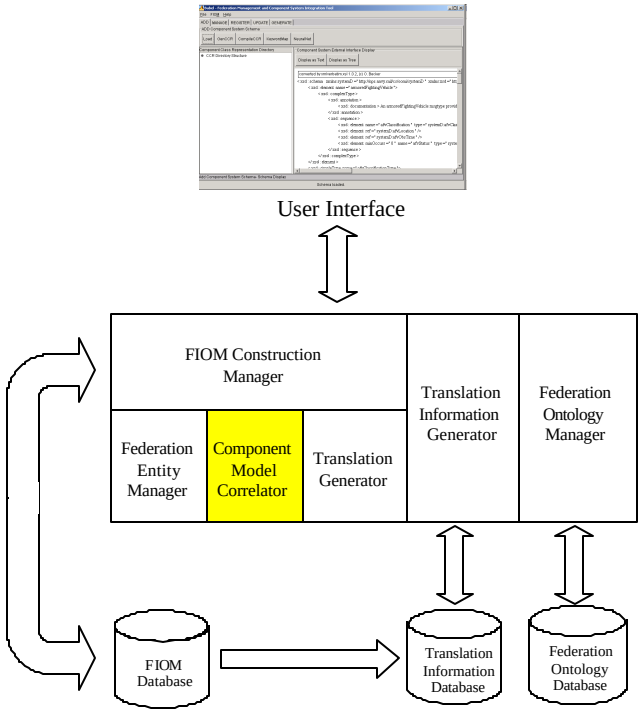


Figure III-2. OOMI IDE Block diagram showing the relationship of the Component Model Correlator to the rest of the OOMI IDE. [after You02]

The component model correlator methodology defined by Young [You02] and Pugh [Pug01] involves a two-phase filtering process using a mix of classical and artificial intelligence approaches. Correlation takes place during the *Register Component Class Representation (CCR)* phase of the overall FIOM construction process within the OOMI IDE. [You02] Phase one is a semantic search based upon the classical approach of keyword matching. The methodology for the semantic search process is discussed in Section III.E. The implementation of a semantic correlator is presented in Section IV.C.2. Phase two uses neural networks to identify syntactic correspondence between CCRs and FCRs through the use of attribute pattern matching. The methodology for the syntactic search process is discussed in Section III.F. The implementation of a syntactic correlator is presented in Section IV.C.3. Each phase produces a correlation score whereby candidate FEVs can be ranked according to best match. The IDE also provides the capability to choose selected FEVs from the first phase of the correlation process for submission to the second phase. The desired effect is that the first phase will narrow the search scope for the computationally expensive neural networks used in the second phase.

A threshold value can be set for both phases of the correlation process to adjust the number of potential matches the interoperability engineer must examined. Although the OOMI IDE provides computer-aid for matching CCRs to the appropriate FE, the interoperability engineer must provide the ultimate determination of whether a CCR and FEV refer to the same real-world entity. [You02] Figure III-3 outlines the correlation methodology.

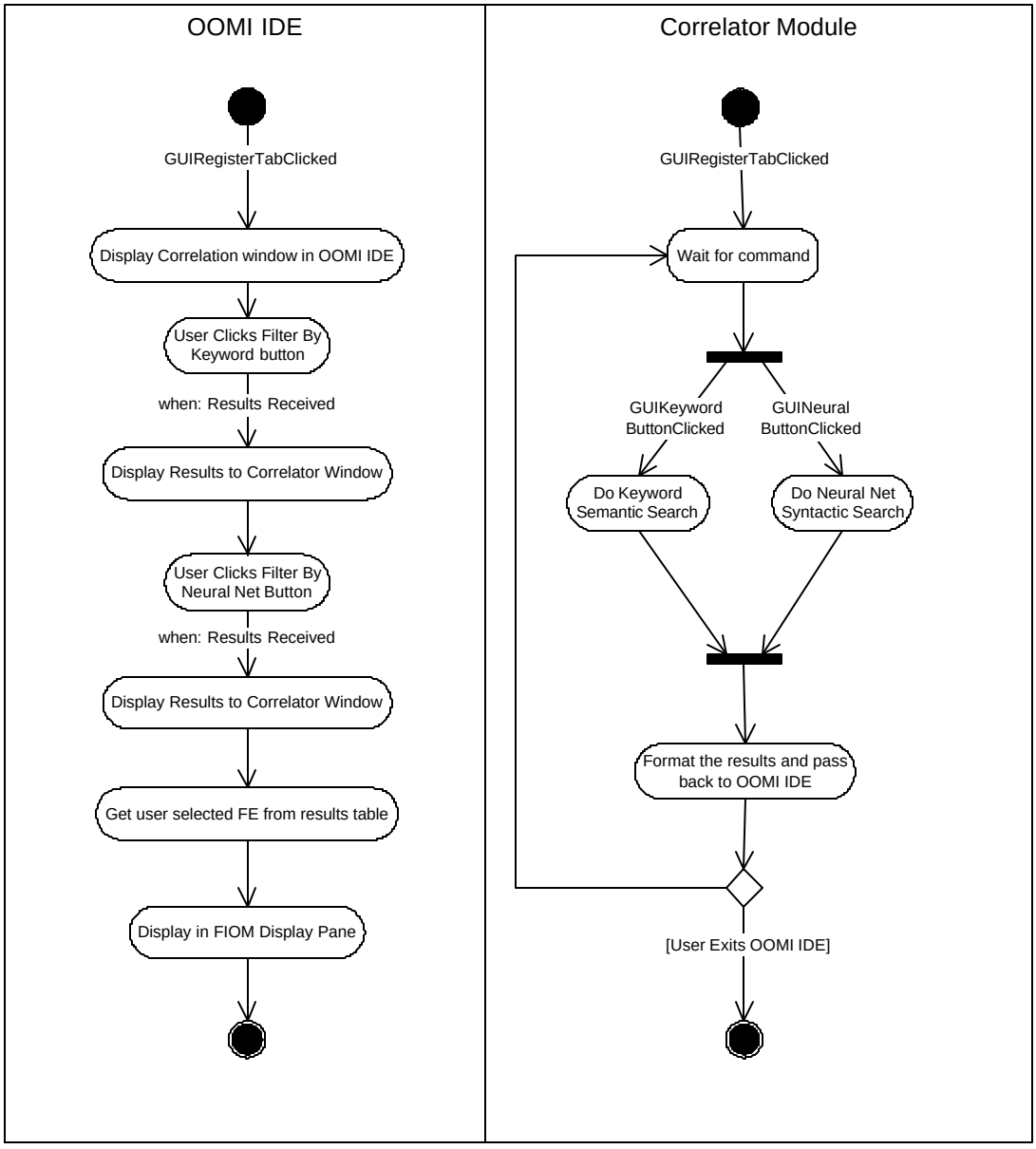


Figure III-3. UML Activity Diagram showing the component model correlation methodology within the OOMI IDE.

In order to utilize the component model correlator, semantic and syntactic search components must first be generated during the *Add Component System External Interface* phase of the overall FIOM construction process. [You02] As stated in the previous paragraph, the semantic search process uses keyword lists to find potential FEVs that match a CCR. A keyword list is generated from an FCR or CCR XML Schema file. The methodology for generating semantic components is discussed in Section III.C. The implementation of a semantic component generator is presented in Section IV.B.2. The syntactic search process attempts to conduct attribute pattern matching between a CCR and FCR through the use of backpropagation neural networks. In order to use this technique, vectors are needed that describe the structure of a CCR attribute or operation and an FCR attribute or operation.

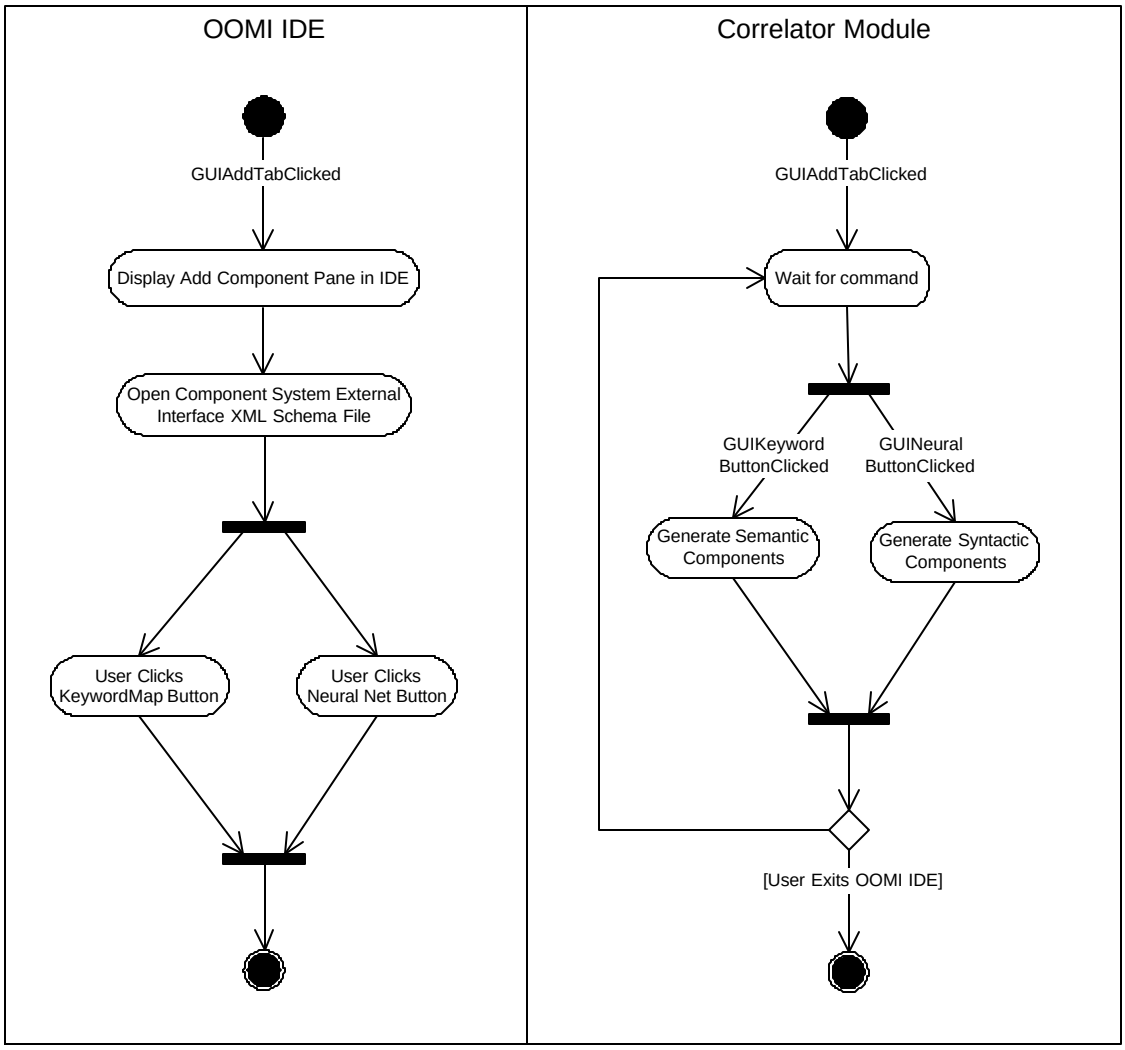


Figure III-4. UML Activity Diagram showing the generation of semantic and syntactic components from within the OOMI IDE.

When an FEV or CCR is added to the FIOM, discriminator vectors are created that describe the structure of each attribute and operation of the class representation. For an FEV, the discriminator vectors are then used to train a backpropagation neural network. Once trained, the neural network can be used to correlate a CCR to an FCR by attribute pattern matching. The methodology used for the generation of the syntactic search components is described in Section III.D. The implementation of a syntactic component generator is presented in Section IV.B.3. The pattern matching is performed by passing each discriminator vector of a CCR through the neural network of an FCR and computing

a pattern match score. Figure III-4 illustrates the process of generating semantic and syntactic components within the OOMI IDE.

An important feature of the OOMI IDE and the correlation methodology is the use of well-formed XML Schemas to represent the information exported or imported through a component system's external interfaces and to provide persistent storage for FCRs. The term "well-formed" means that the XML Schema is syntactically correct according to the W3C's XML specification. [BM01] Thus, an ill-formed document will not be accepted for processing. This simplifies the internal code of parsers and also speeds up the processing of documents. An XML Schema is used to define the physical structure of an FCR or CCR through the use of specific tags to represent cardinality, relationships and data types. Complex data types can be broken down to simple types and characteristics such as enumeration values can be defined. Further, an XML Schema can include descriptor fields that provide comments about the overall schema, individual attributes or operations. Specific XML implementation considerations are discussed in Section IV.B.1.

For purposes of discussion, the scenario presented in Figure II-1 of Section II.A.3 will be used throughout the following sections. In the diagram, four systems are shown, each providing a model for a real-world entity called a ground combat vehicle. System A and B have similar models of the real-world entity and constitute view 1 of the ground combat vehicle. System C provides view 2 of the ground combat vehicle and System D provides view 3. Figure II-8 shows an FE representation of the ground combat vehicle as it would be modeled in the FIOM.

C. SEMANTIC COMPONENT GENERATION

In order to perform the semantic search phase of component model correlation, the semantic search algorithm requires two list of keywords: 1) a keyword list describing the candidate Component Class Representation (CCR) and, 2) a keyword list describing the Federation Class Representation (FCR) of a real-world entity. The semantic component generation process is illustrated in Figure III-5. Keyword information for a CCR is included in the XML Schema used to characterize a single component system's external interface. For a federation representation (i.e. an FCR describing an FEV), keyword information can be obtained from an FCR XML Schema extracted from the FIOM, or can

be directly entered by the interoperability engineer. Table III-1 defines a list of XML schema fields from which keyword descriptions can be obtained. Two additions are made to Pugh's initial list: the *ref* attribute for an *xsd:element* component and the *name* attribute of an *xsd:simpleType* component. The *ref* attribute is similar in purpose to the *type* attribute, providing a description of the underlying data type names. The *simpleType name* attribute also gives the names of the underlying data types defined in the schema. The addition of these fields provides slightly more keyword detail than the original field list provided by Pugh.

Table III-1. XML Schema fields used for keyword generation. [after Pug01]

Field	Attribute	Details
xsd:element	"name"	The name attribute typically equates to the field name used in the underlying database.
xsd:element	"type"	For schemas using global types. This attribute's value is usually descriptive of the kind of data in the subtype. (e.g. "data type")
xsd:element	"ref"	This attribute value is also descriptive of the kind of data in the subtype. (e.g. "data type")
xsd:documentation	N/A	The text in this element is the "description" field from the data dictionary. It is typically a human-readable free text explanation of the field's use or format.
xsd:simpleType	"name"	Gives the name of the underlying data types. Particularly beneficial if the data types declared in the element tags have the target namespace appended to the front of the name.
xsd:attribute	"name"	Gives amplifying information about a simple or complex type.
xsd:enumeration	'value"	Used to constrain the values of types. Usually used to limit a message field to several values, which will reveal the use of the message (e.g.. "SUB", "SURF", "AIR")

Keyword information for a CCR is generated during the *Add Component System External Interface* phase of FIOM construction from the XML Schema defining the CCR. As proposed by Young, the keyword information is not extracted directly from the XML Schema document. [You02] Instead, keywords are extracted using the *CCRSchema* component created in the *Add Component System External Interface* process. Recall from

Section II.D.4 that the *CCRSchema* interface is a component of the FIOM Framework and contains the complete information on the attributes of the CCR. The keyword list created by the keyword generator is then stored with the CCR as a *CCRSemantics* component.

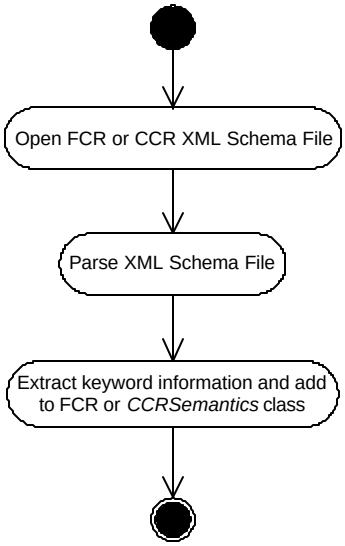


Figure III-5. UML activity diagram of the component model correlator semantic component generation process.

Keyword information for an FCR is generated in the *Manage Federation Entities* phase of the OOMI IDE either from an XML Schema defining the FCR, from information about a real-world entity entered by the interoperability engineer, from keywords entered by the interoperability engineer, or from a combination of the three. As with the generation of CCR keyword information, the keyword information for an FCR is not extracted directly from an XML Schema document. [You02] Instead, keywords are extracted using the *FCRSchema* component created in the *Manage Federation Entities* phase. The keyword list created by the keyword generator is then stored with the FCR as an *FCRSemantics* component.

This thesis proposes a modification to the methodology described above in order to simplify the extraction of keywords. Rather than extracting keywords from the *FCRSchema* or *CCRSchema* components, it is proposed that the keywords be extracted straight from an XML Schema. The reasoning for this change is to rely on the lightweight

standards and support for XML rather than create a complex process based on Java reflection for keyword generation.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/systemD'
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:systemD="http://nps.navy.mil/cs/oomi/systemD"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:element name="armoredFightingVehicle">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation>The armoredFightingVehicle of System D models the ground combat
 vehicle real-world entity.</xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="afvClassification" type="systemD:afvClassificationType"/>
                <xsd:element ref="systemD:afvLocation"/>
                <xsd:element ref="systemD:afvObsTime"/>
                <xsd:element name="afvStatus" type="systemD:afvStatusType" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="afvClassificationType">
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="5"/>
            <xsd:maxLength value="14"/>
            <xsd:enumeration value="battleTank"/>
            <xsd:enumeration value="rocketLauncher"/>
            <xsd:enumeration value="truck"/>
            <xsd:enumeration value="unknown"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="afvLocation">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation/>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="latitude" type="systemD:latitudeType"/>
                <xsd:element name="longitude" type="systemD:longitudeType"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="afvObsTime">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation>The day of a month and timekeeping in hours and minutes of a calendar
 day, using the 24-hour clock system referenced to Greenwich Mean Time (GMT).</xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="day" type="systemD:dayType"/>
                <xsd:element name="hourTime" type="systemD:hourTimeType"/>
                <xsd:element name="minuteTime" type="systemD:minuteTimeType"/>
                <xsd:element name="stdTimeZone" type="systemD:stdTimeZoneType"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

        .
        .
        .

</xsd:schema>
```

Figure III-6. Partial, well-formed XML Schema for the armored fighting vehicle.

Young's methodology relies on the use of Java-XML Data-Binding, a process in which an XML document is converted to its equivalent representation as a Java object and vice versa. During the *Add Component System External Interface* and *Manage Federation Entities* phases, XML schemas are unmarshalled, or converted to Java class objects. The *FCRSchema* and *CCRSchema* components, from which keyword information is supposed

to be generated, are then created. For persistent storage, when the user closes the OOMI IDE, the Java objects in the FIOM, including the *FCR* and *FCRSchema* components, are marshalled, or converted to XML Schema files. [Lee02]

The proposed change to the methodology capitalizes on the lightweight nature of XML Schemas and the open-source Java packages available for XML processing. The start point of the *Add Component System External Interface* phase is a well-formed XML Schema. Figure III-6 shows a partial, well-formed XML schema for the armored fighting vehicle modeled by system D of the chapter example. Using the W3C Document Object Model (DOM) or the Apache Project JDOM package, keywords can be quickly generated straight from the XML Schema without the overhead required by Java-XML binding. A partial list of keywords generated for the armored fighting vehicle can be seen in Figure III-7 in the form of an XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<keywords>armoredFightingVehicle The armoredFightingVehicle of
System D models the ground combat vehicle real-world entity
afvClassification systemD:afvClassificationType systemD:afvLocation
systemD:afvObsTime afvStatus systemD:afvStatusType
afvClassificationType battleTank rocketLauncher truck unknown
afvLocation latitude systemD:latitudeType longitude
systemD:longitudeType afvObsTime The day of a month and timekeeping
in hours and minutes of a calendar day, using the 24-hour clock system
referenced to Greenwich Mean Time (GMT) day systemD:dayType
hourTime systemD:hourTimeType minuteTime systemD:minuteTimeType
stdTimeZone systemD:stdTimeZoneType
</keywords>
```

Figure III-7. Partial list of keywords generated with the semantic component generator.

Picking up with the original methodology, the generated keyword list is stored with the CCR as a *CCRSemantics* component. Similarly, for the generation of an FCR keyword list, the FCR's XML Schema can be parsed during the *Manage Federation Entities* phase. Another benefit of extracting keywords directly from an XML Schema is the decoupling of the keyword generator from the FIOM Framework. With an XML Schema as the only input, a keyword generator must simply analyze the XML Schema and return a list of keywords. The XML Schema elements listed in Table III-1 are standardized by the W3C and are logical choices for keyword extraction. By decoupling the keyword generator from the FIOM Framework, it becomes possible to use commercial

XML Schema keyword extractors should they become available. Additionally, changes to either the FIOM Framework or the implementation of a keyword generator will not adversely affect each other.

D. SYNTACTIC COMPONENT GENERATION

Syntactic components are used to support the neural network based syntactic matching process in the component model correlator [You02]. The syntactic components include two subcomponents. The first subcomponent is the discriminator vectors that describe the structure of the attributes of an entity defined by an FCR or CCR. The second subcomponent is a trained neural network belonging to an FCR. A description of these components as well as the methodology for their generation is discussed below. Figure III-8 illustrates the syntactic component generation process.

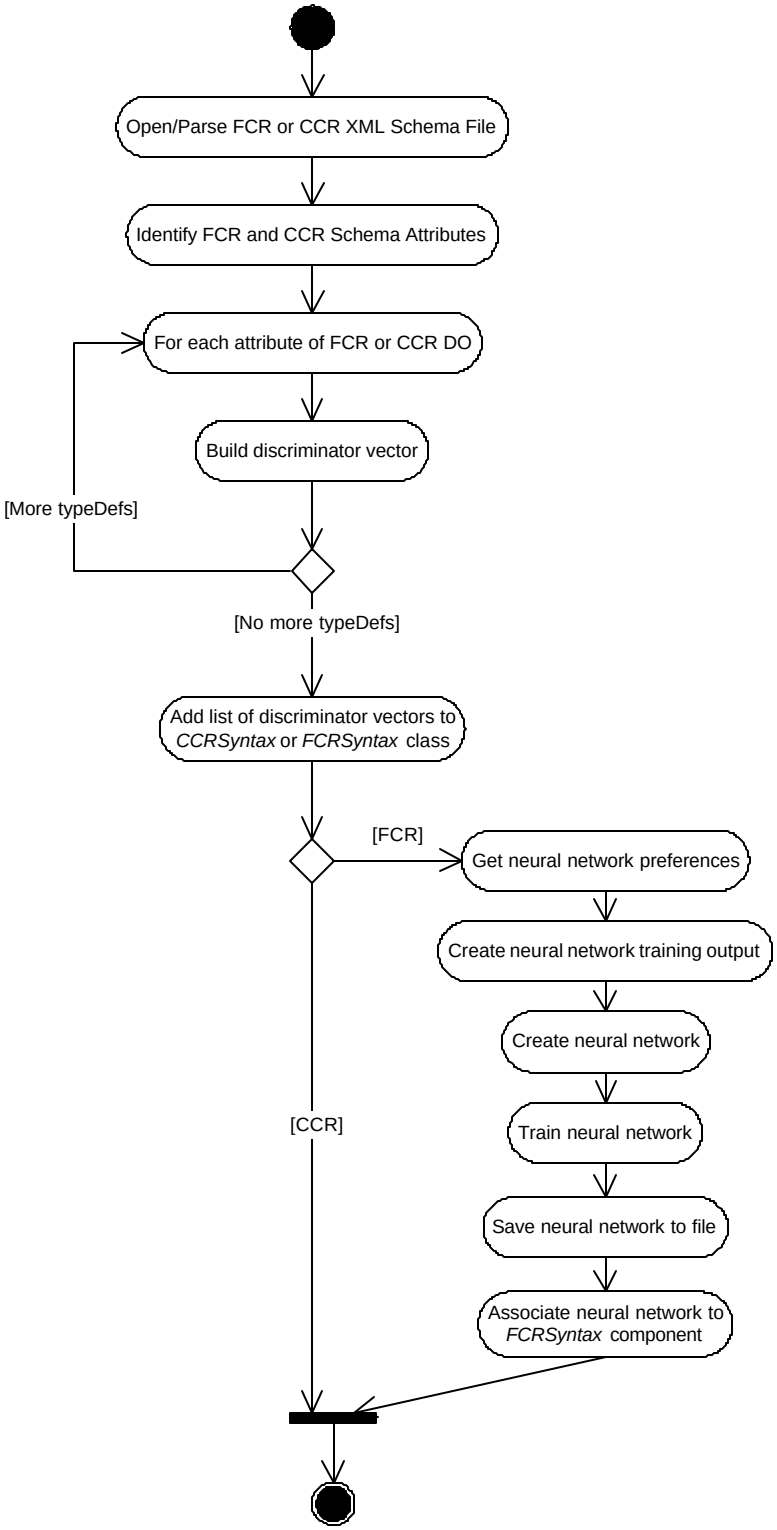


Figure III-8. UML activity diagram of the component model correlator syntactic component generation process.

1. Discriminator Vector Generation

A backpropagation neural network is used to perform syntactic attribute pattern matching between a CCR and an FCR. Syntactic pattern matching refers to matching the syntax of attributes, or the way in which attributes are structurally represented. Some examples of syntactic information about an attribute include the attribute's data type, whether or not the attribute is required, the maximum value if the attribute is a number, or the length if the attribute is a string. In order to use a backpropagation neural network, the neural network must first be trained by the backpropagation supervised learning algorithm. As discussed in Section III.A.4, the learning algorithm requires a set of input vectors with corresponding desired output vectors. When performing pattern matching with a trained neural network, the input to the network is a set of candidate input vectors. In the OOMI IDE component model correlator methodology, the vectors used for the neural networks are called discriminator vectors. A discriminator vector is an array of values in the range [0.0, 1.0] used to represent the data content and structure of each attribute and operation in a CCR or FCR. [You02]

Discriminator Vectors for a CCR are created during the *Add Component System External Interface* phase of the OOMI IDE from an XML Schema of a single component system external interface. Discriminator vectors for an FCR are created during the *Manage Federation Entities* phase of the OOMI IDE. During the vector generation process, syntactic information is extracted from the XML Schema defining the CCR or FCR and a discriminator vector is constructed for each attribute and operation. A discriminator vector schema is defined in Table III-2. The discriminator vector schema proposed benefits from the metadata contained within an XML Schema. For example, the type specification of the discriminator vector is extracted from the *base type definition* component of an XML Schema *simple type definition*, which is a standardized XML Schema component from [BM01]. Additionally, the *minOccurs* and *maxOccurs* elements of the discriminator vector are populated from the matching *particle schema components* of an XML *element* as defined in [TBM+01]. Lastly, the discriminator vector elements *minLength, maxLength, totalDigits, fractionDigits, pattern, enumeration, maxInclusive,*

maxExclusive, *minExclusive*, and *minInclusive* are taken from the corresponding *constraining facets* of an XML data type, which are specified in [BM01].

Table III-2. Metadata Based Discriminators Used in Syntactic Correlation Process.

Number	Discriminator	Description
		Structural Information
1	propertyType	Operation or Attribute
2	isComplex	Describes whether an attribute is complex or atomic
3	numSubtypes	If attribute is complex, number of subtypes
4	numReqdSubtypes	If attribute is complex, number of required subtypes
5	numOptSubtypes	If attribute is complex, number of optional subtypes
6	numOperations	For complex attribute – total no. of operations defined for type
7	numParameters	For operation – number of parameters For complex attribute – Sum of parameters for all operations defined for that attribute and any subtypes
		Type Specifications
8	string type	java.lang.String type
9	boolean type	primitive Boolean type
10	float type	primitive float type
11	double type	primitive double type
12	bigDecimal type	java.math.BigDecimal type
13	int type	primitive int type
14	long type	primitive short type
15	short type	primitive short type
16	other type	type other than listed above
		For Atomic Attributes – Specify the data type For Complex Attributes – sum of the number of subtypes of each type For Operations – sum of the number of parameters of each type
		Frequency of Occurrence
17	minOccurs	minimum number of times attribute must occur in class modeling real-world entity
18	maxOccurs	maximum number of times attribute may occur in class modeling real-world entity
		Data Size Specification
19	minLength	For Atomic String Type Attribute – minimum length of string For Complex Attribute– Sum of minLengths of all string subtypes
20	maxLength	For Atomic String Type Attribute – maximum length of string For Complex Attribute – Sum of maxLengths for all string subtypes
21	totalDigits	For Atomic bigDecimal Type Attribute – total number of digits included in attribute For Complex Attribute – Sum of total number of digits for all bigDecimal types
22	fractionDigits	For Atomic bigDecimal Type Attribute– number of digits in fraction part of attribute For Complex Attribute – Sum of total number of digits in fraction part for all bigDecimal types
		Data Value Constraints
23	pattern	For Atomic Attribute – restriction to values allowed for string and numeric types For Complex Attribute – number of attributes with pattern defined
24	numEnumerations	For Atomic Attribute – number of enumeration values for attribute For Complex Attribute – sum of number of enumeration values for all subtypes
25	minExclusive	lower open bound of interval defined for numeric attribute types
26	maxExclusive	upper open bound of interval defined for numeric attribute types
27	minInclusive	lower closed bound of interval defined for numeric attribute types
28	maxInclusive	upper closed bound of interval defined for numeric attribute types

Each element of a CCR or FCR is first evaluated to determine if it is an attribute or an operation. For an operation, only the *propertyType*, *numParameters*, and type

specification discriminators are used. The discriminator *propertyType* simply identifies whether the vector describes an operation or an attribute. The discriminator *numParameters* represents the number of input parameters required by the operation's signature. The type specification discriminators for an operation represent a tally of the number of each data type that is a parameter in the operation. For example, an operation that takes two String parameters would have the value "2" stored in the *stringType* discriminator element.

If the element is an attribute, all discriminators, except for the discriminator *numParameters*, can be utilized. An attribute is further categorized as complex or atomic. An atomic attribute is one that cannot be decomposed into simpler types. The discriminator elements for an atomic attribute describe its basic structure. As shown in Table III-2, the discriminators for a complex element represent an aggregation of its atomic elements. To illustrate, suppose, a complex element is composed of two atomic elements of type String. The aggregation represented by the complex element discriminator vector would indicate the value "2" for the *numSubTypes* discriminator and the value "2" for the *stringType* discriminator. If the String subtypes had values for *maxLength* of 15 and 20 respectively, the aggregate value of maxLength for the complex element would be "35." The technique of providing an aggregate discriminator vector for a complex element can be used throughout a type hierarchy. It is expected to provide adequate distinction between CCR and FCR attributes to enable pattern matching.

The backpropagation neural networks in the OOMI IDE syntactic correlator use Eq 3.1, the binary sigmoid function, for node activation. As stated in Section III.A.4, Eq 3.1 is the most typical activation function and has the range [0.0,1.0]. As such, all values in the input vector are required to be in the range [0.0, 1.0]. In order to satisfy this requirement, an algorithm must be provided to normalize the raw values of the discriminator elements into the range [0.0, 1.0]. Table III-3 and Table III-4 define the normalized discriminator values used in the syntactic correlation process.

Table III-3. Discriminator Values Used for Syntactic Correlation.

Number	Discriminator	Value to Vector
		Structural Information
1	propertyType	Operation – 0.0 Attribute – 1.0
2	isComplex	If yes – 1.0 Otherwise – 0.0
3	numSubtypes	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] Note 1
4	numReqdSubtypes	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] Note 1 (subtype required unless minOccurs = 0)
5	numOptSubtypes	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] Note 1 (subtype optional only if minOccurs = 0)
6	numOperations	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] Note 1
7	numParameters	For atomic attribute – 0.0 For operation – value for # of parameters normalized to [0.0, 1.0] Note 1 For complex attribute – value for sum of parameters for all operations in subtype normalized to [0.0, 1.0] Note 1
		Type Specifications
8	string type	If Atomic – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
9	boolean type	If Atomic – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
10	float type	If Atomic – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
11	double type	If Atomic – < 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
12	bigDecimal type	If Atomic – < 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
13	int type	If Atomic – < 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
14	long type	If Atomic – < 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
15	short type	If Atomic – < 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
16	other type	If Atomic – < 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If Complex – Value normalized to [0.0, 1.0] Note 1
		Frequency of Occurrence
17	minOccurs	If Optional – 0.0 Otherwise – 1.0
18	maxOccurs	If not specified – raw value = 1, normalized to [0.0, 1.0] Note 1 Otherwise - value normalized to [0.0, 1.0] Note 1

Note 1: Uses Eq 3.4: $f(x) = 2 * (1/(1 + k\text{-}x) – 0.5)$ with $k = 1.01$

Table III-4. Discriminator Values Used for Syntactic Correlation (Continued).

Number	Discriminator	Value to Vector
		Data Size Specification
19	minLength	If operation, attribute not string, or minLength not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 1
20	maxLength	If operation, attribute not string, or maxLength not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2
21	totalDigits	If operation, attribute not bigDecimal, or totalDigits not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2
22	fractionDigits	If operation, attribute not bigDecimal, or fractionDigits not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2
		Data Value Constraints
23	pattern	If Atomic – If pattern defined – 1.0 Else 0.0 For Complex Attribute – Value normalized to [0.0, 1.0] Note 1
24	numEnumerations	If operation or numEnumerations not specified – 0.0; Otherwise, value normalized to [0.0, 1.0] Note 1
25	minExclusive	If operation, attribute not numeric, or minExclusive not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2
26	maxExclusive	If operation, attribute not numeric, or maxExclusive not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2
27	minInclusive	If operation, attribute not numeric, or minInclusive not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2
28	maxInclusive	If operation, attribute not numeric, or maxInclusive not specified – 0.0; Otherwise - value normalized to [0.0, 1.0] Note 2

Note 1: Uses Eq 3.4: $f(x) = 2 * (1/(1 + k^{-x}) - 0.5)$ with $k = 1.01$
Note 2: Uses Eq 3.6: $f(x) = \log(x + 1)/5$

For some discriminators, such as *minOccurs* and *pattern* (for atomic attributes), the Boolean nature of the discriminator enables a direct mapping to a 0 or 1 value. Other parameters, such as *numSubtypes*, *numReqdSubtypes*, *numOptSubtypes*, *numOperations*, *numParameters*, *maxOccurs*, *minLength*, *maxLength*, *totalDigits*, *fractionDigits*, *numEnumerations*, *minExclusive*, *maxExclusive*, *minInclusive*, *maxInclusive*, and the type specifications (for complex attributes) must be normalized to a value in the range of [0.0, 1.0]. For these values Li and Clifton used a SIGMOID-like function in order to avoid false matches or false drops that could occur when using a linear normalization function. For positive numeric values in the range of [0.0, 100.0] Li and Clifton used the function

$$f(x) = 2 \cdot \left(\frac{1}{1 + k^{-x}} - 0.5 \right) \qquad \text{[Eq 3.4]}$$

where *k* is a constant equal to 1.01 and *x* is the parameter value to be normalized [LC00]. For other numeric values that may be positive or negative [-50.0, 50.0], they used the function

$$f(x) = \frac{1}{1 + k^{-x}} \qquad \text{[Eq 3.5]}$$

with *k* = 1.01. As each of the parameters *numSubtypes, numReqdSubtypes, numOptSubtypes, numOperations, numParameters, maxOccurs, minLength,* and *numEnumerations* are expected to have values in the range [0.0, 100.0], Eq 3.4 is used to normalize the values if these discriminators. The parameters *maxLength, totalDigits,* and *fractionDigits* may have values that exceed 100 for complex attributes. For example, a CCR attribute describing a data type for representing altitude, with a base type of integer or long, could have an extremely high value for *maxLength.* To account for this situation, a third function

$$f(x) = \frac{\log(x + 1)}{5} \qquad \text{[Eq 3.6]}$$

has been defined which provides adequate discrimination between values in the range of [0, 100000]. Similarly, while parameters *minExclusive, maxExclusive, minInclusive,* and *maxInclusive* may have either positive or negative values, suggesting the use of Eq 3.2 for normalizing their values, the potential to exceed the bounds of [-50.0, 50.0] prescribed for use of that equation has led to the use of Eq 3.6 for normalizing these parameters as well. Figure III-9 provides a comparison graph showing the limits of each normalization equation with respect to the range [0.0,1.0].

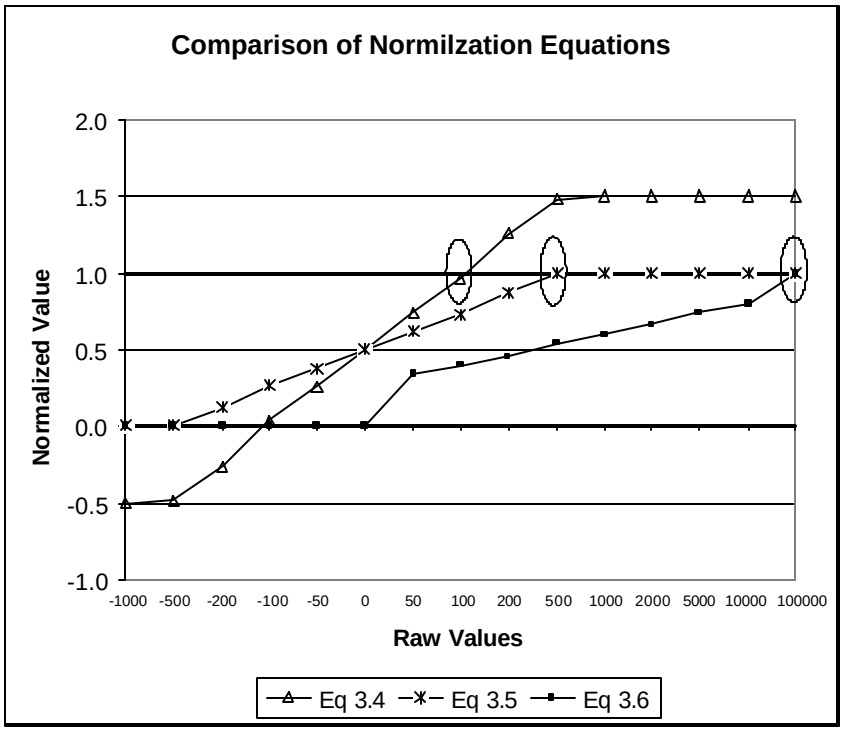


Figure III-9. Comparison of normalization equations.

Lastly, as Table III-3 shows, the discriminator values for an atomic attribute's data type require vectors of binary values instead of values within the range [0.0,1.0]. Establishing a categorical value in the range [0.0,l.0] for each data type would construe that a data type is "closer to" one type than another. For example if *float* were represented by 0.1, *string* by 0.2, and *bigDecimal* by 0.3, the neural network would conclude that a *float* is more like a *string* than a *bigDecimal*. Clearly, this does not have the desired effect. Instead, each data type has a sub vector with a unique value pattern. For example, *string* is represented by the vector

< 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 >,

boolean by

< 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 >

and so on. With this technique, each element of the type sub vector will stimulate its own neuron in the neural network and avoid any perceived "closeness" between data types.

As with the generation of the semantic components discussed in Section III.C, this thesis proposes a change to the methodology proposed by Young [You02]. In Young's methodology for generating syntactic components, the discriminator generator uses the *FCRSchema* and *CCRSchema* components of the FIOM Framework as the source for syntactic information. [You02] Rather than extract syntax information from the *FCRSchema* or *CCRSchema* components, it is proposed that the syntactic information be extracted directly from an XML Schema. Figure III-10 shows the values for the discriminator vectors created from the armored fighting vehicle XML schema of Figure III-6. Again, the reasoning for this change is to rely on the lightweight standards and support for XML and decouple the discriminator generation from the FIOM Framework. As shown, most of the elements of a discriminator vector have a direct mapping to XML Schema components. This proposed change leaves intact the process of storing the generated discriminator vectors with a CCR as a *CCRSyntax* component or with an FCR as an *FCRSyntax* component.

ArmoredFightingVehicle

Construction of Discriminator Vector

Normalization equation		None	None	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4
Attribute		Oper or Attribute	is Complex	Num Subtypes	Num Required Subtypes	Num Optional Subtypes	Num Opers	Num Total Params	For atomic attributes - specify the type For complex attributes - sum of the number of suptypes or each type								
									string	boolean	float	double	big Decimal	int	long	short	other
afvClassification	Raw	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	Vector	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
afvLocation	Raw	1	1	6	2	4	0	0	4	0	0	0	0	0	0	0	0
	Vector	1.0	1.0	0.52984	0.50995	0.51990	0.5	0.5	0.51990	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
afvObsTime	Raw	1	1	4	0	4	0	0	4	0	0	0	0	0	0	0	0
	Vector	1.0	1.0	0.51990	0.5	0.5	0.5	0.5	0.51990	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
afvStatus	Raw	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	Vector	1.0	0.0	0.5	0.5	0.5	0.5	0.5	1.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	Raw																
	Vector																

Construction of Discriminator Vector (Continued)

Normalization equation		None	3.4	3.4	3.6	3.6	3.6	3.4	3.4	3.6	3.6	3.6	3.6
Attribute		min Occurs	max Occurs	min Length	max Length	total Digits	fraction Digits	pattern	Num Enums	min Exclusive	max Exclusive	min Inclusive	max Inclusive
afvClassification	Raw	0	1	5	14	0	0	0	4	0	0	0	0
	Vector	0.0	0.50498	0.52487	0.23522	0.0	0.0	0.0	0.51990	0.0	0.0	0.0	0.0
afvLocation	Raw	1	15	15	15	0	0	0	4	0	0	0	0
	Vector	1.0	0.57449	0.57449	0.24082	0.0	0.0	0.5	0.51990	0.0	0.0	0.0	0.0
afvObsTime	Raw	0	5	7	7	0	0	4	0	0	0	0	0
	Vector	0.0	0.52487	0.53481	0.18062	0.0	0.0	0.51990	0.5	0.0	0.0	0.0	0.0
afvStatus	Raw	0	3	7	11	0	0	0	3	0	0	0	0
	Vector	0.0	0.51492	0.53481	0.21584	0.0	0.0	0.0	0.51492	0.0	0.0	0.0	0.0
	Raw												
	Vector												

Resulting Neural Net Input Discrimintator Vectors and Output Training Vectors

Attribute	Input Discriminator Vector
afvClassification	1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.50498 0.52487 0.23522 0.0 0.0 0.0 0.51990 0.0 0.0 0.0 0.0
afvLocation	1.0 1.0 0.52984 0.50995 0.51990 0.5 0.5 0.51990 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 1.0 0.57449 0.57449 0.24082 0.0 0.0 0.5 0.51990 0.0 0.0 0.0 0.0
afvObsTime	1.0 1.0 0.51990 0.5 0.5 0.5 0.5 0.51990 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.0 0.52487 0.53481 0.18062 0.0 0.0 0.51990 0.5 0.0 0.0 0.0 0.0
afvStatus	1.0 0.0 0.5 0.5 0.5 0.5 0.5 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.0 0.51492 0.53481 0.21584 0.0 0.0 0.0 0.51492 0.0 0.0 0.0 0.0

Figure III-10. Construction of the discriminator vectors for the armoredFightingVehicle CCR.

2. Neural Network Generation

Every FCR in the FIOM must have a trained neural network as one of its syntactic components. As shown in Figure III-8, four major steps are involved in the generation of an FEV FCR neural network. The first step is to create the desired output vectors for the network. The second step is to create an instance of a backpropagation neural network. Next the newly created network is trained. Lastly, the trained neural network is saved to a file and the file name is stored with the FCR in the *FCRSyntax* component of the FIOM Framework.

The first required step in generating the neural network is the creation of the desired output vectors needed to calculate the output error of the network during training. The output vectors are created with binary values in order to distinguish between each

attribute of the FCR. For example, an FCR with four attributes would have the following desired output vectors:

attribute 1	< 1.0, 0.0, 0.0, 0.0 >
attribute 2	< 0.0, 1.0, 0.0, 0.0 >
attribute 2	< 0.0, 0.0, 1.0, 0.0 >
attribute 2	< 0.0, 0.0, 0.0, 1.0 >

The length of the desired output vectors equals the number of attributes represented by the FCR. The vector for each attribute contains a single value of 1.0 with the remaining values set to 0.0. The position of the value 1.0 creates a unique binary pattern for each attribute. The construction of the discriminator vectors and desired output vectors for the groundCombatVehicle_View3 FCR is shown in Figure III-11. Creating the output vectors with this technique facilitates pattern matching when CCR attributes are passed through an FCR neural network.

In the second step, an instance of a backpropagation neural network is created. The component model correlator uses a three-layer backpropagation neural network. As discussed in Section III.A.4, this type of neural network has an input layer of nodes, a hidden layer of nodes, and an output layer of nodes. In order to create the neural network, the number of nodes in each layer must be specified. The number of input layer nodes is equal to the number of discriminator elements defined in Table III-2, which is equal to 28. The number of output layer nodes equals the number of attributes of the FEV FCR. This provides one distinct node for the representation of each FCR attribute and matches the desired output vector standard described above. The number of nodes in the hidden layer is

$$numHidden = \left\lfloor \frac{numInput + numOutput}{2} \right\rfloor \qquad \text{[Eq 3.7]}$$

which is the floor of the average of the input nodes and output nodes. The weight matrices of the neural network are automatically generated to fully connect the network graph between the input layer and the output layer.

GroundCombatVehicle_View3

Construction of Discriminator Vector

Normalization equation		None	None	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4	3.4
Attribute		Oper or Attribute	is Complex	Num Subtypes	Num Required Subtypes	Num Optional Subtypes	Num Opers	Num Total Params	For atomic attributes - specify the type For complex attributes - sum of the number of suptypes or each type								
									string	boolean	float	double	big Decimal	int	long	short	other
gCV3_VehicleType	Raw	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	Vector	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
gCV3_Position	Raw	1	1	6	0	6	0	0	4	0	0	0	0	0	0	0	0
	Vector	1.0	1.0	0.52984	0.50000	0.52984	0.5	0.5	0.51990	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
gCV3_Time	Raw	1	1	4	0	4	0	0	4	0	0	0	0	0	0	0	0
	Vector	1.0	1.0	0.51990	0.5	0.5	0.5	0.5	0.51990	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
gCV3_StatusType	Raw	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	Vector	1.0	0.0	0.5	0.5	0.5	0.5	0.5	1.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	Raw																
	Vector																

Construction of Discriminator Vector (Continued)

Normalization equation		None	3.4	3.4	3.6	3.6	3.6	3.4	3.4	3.6	3.6	3.6	3.6
Attribute		min Occurs	max Occurs	min Length	max Length	total Digits	fraction Digits	pattern	Num Enums	min Exclusive	max Exclusive	min Inclusive	max Inclusive
gCV3_VehicleType	Raw	0	1	5	16	0	0	0	6	0	0	0	0
	Vector	0.0	0.50498	0.52487	0.24609	0.0	0.0	0.0	0.52984	0.0	0.0	0.0	0.0
gCV3_Position	Raw	0	6	15	15	0	0	0	4	0	0	0	0
	Vector	0.0	0.52984	0.57449	0.24082	0.0	0.0	0.5	0.51990	0.0	0.0	0.0	0.0
gCV3_Time	Raw	0	5	7	7	0	0	4	0	0	0	0	0
	Vector	0.0	0.52487	0.53481	0.18062	0.0	0.0	0.51990	0.5	0.0	0.0	0.0	0.0
gCV3_StatusType	Raw	0	1	7	11	0	0	0	3	0	0	0	0
	Vector	0.0	0.50498	0.53481	0.21584	0.0	0.0	0.0	0.51492	0.0	0.0	0.0	0.0
	Raw												
	Vector												

Resulting Neural Net Input Discrimintator Vectors and Output Training Vectors

Attribute	Input Discriminator Vector	Desired Output
gCV3_VehicleType	1.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.50498 0.52487 0.23522 0.0 0.0 0.0 0.51990 0.0 0.0 0.0 0.0	1.0 0.0 0.0 0.0
gCV3_Position	1.0 1.0 0.52984 0.50995 0.51990 0.5 0.5 0.51990 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 1.0 0.57449 0.57449 0.24082 0.0 0.0 0.5 0.51990 0.0 0.0 0.0 0.0	0.0 1.0 0.0 0.0
gCV3_Time	1.0 1.0 0.51990 0.5 0.5 0.5 0.5 0.51990 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.0 0.52487 0.53481 0.18062 0.0 0.0 0.51990 0.5 0.0 0.0 0.0 0.0	0.0 0.0 1.0 0.0
gCV3_StatusType	1.0 0.0 0.5 0.5 0.5 0.5 0.5 1.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.0 0.51492 0.53481 0.21584 0.0 0.0 0.0 0.51492 0.0 0.0 0.0 0.0	0.0 0.0 0.0 1.0

Figure III-11. Construction of the discriminator vectors and desired output vectors for the GroundCombatVehicle_View3 FEV FCR.

The neural network is trained in the third step with the backpropagation supervised learning algorithm discussed in Section III.A.4. The generated discriminator vectors and desired output vectors are used for the training process. In a single training instance, an FEV FCR attribute is forwardpropagated through the network and an error is computed between the actual output vector and the desired output vector. The weight matrices of the network are then adjusted. The next attribute is then passed through the network, the output error is calculated and the weights are again adjusted. This process continues until the neural network can produce the correct output, within a preset error tolerance, for every input vector of the training set. The preset error tolerance is set using the "preferences" option in the component model correlator window of the OOMI IDE. A typical value for the error tolerance is 0.1. Usually, neural network training can take

hundreds or even thousands of epochs, where an epoch is one cycle, or one pass of the attribute input vectors through the backpropagation learning algorithm. Once trained, whenever the network sees the discriminator pattern for attribute "x" as the input, it will produce an output vector that resembles the predetermined output pattern "y" used in the training process.

Two other preferences can be set to control the training process: the learning rate and maximum number of epochs. The learning rate is a value in the range [0.0,1.0] and used in the learning algorithm to compute the increment of adjustment to the values of the weight connectors between nodes. Recall from Section III.A.4 that the backpropagation learning algorithm computes the output error of the network and then backpropagates through the network to adjust the values of the weight connectors. The adjustments to the weight values are performed in calculated increments. The lower the learning rate value, the lower the increment. This results in a better-trained neural network, but the time required to train the network, given the small increments in weight adjustments, increases dramatically. The higher the learning rate, the larger the increment, which decreases the time required to train the neural network and increases the possibility of random errors. The maximum number of epochs is a simple maximum limit on the number of epochs the training process can go through. This serves to bound the amount of time taken to train the neural network.

Once trained, the neural network is stored in a formatted file. The implementation in Section IV.D uses a special format for the neural network file as defined in the Java package mwa.ai.neural from [Wat97]. However, an XML document could also be used. The file name is then stored with the FCR in the *FCRSyntax* object.

E. SEMANTIC CORRELATION

The first phase of the component model correlator is the semantic, or keyword matching search. This phase of correlation attempts to narrow the search space for the second phase, the syntactic search discussed in Section III.F. In the semantic search process, a semantic correlator compares the keyword list of a candidate CCR to the keyword list of each FEV FCR contained in the FIOM. The keyword matching process computes a percentage of relevance with the function

$$\%\ relevance = \frac{number\ of\ correct\ matches}{total\ number\ of\ CCR\ keywords} \qquad \text{[Eq 3.7]}$$

Before initiating the semantic correlator, two steps must be performed. First, the user can set the semantic correlation threshold to the desired value or accept the default value set by the OOMI IDE. The semantic correlator will then return FEV FCR matches with a percent relevance above the threshold setting. This is a valuable tool to limit the FEVs the interoperability engineer must analyze. In the second step, the user must select the CCR for which a match is desired. The selected CCR is then termed the candidate CCR.

Once the threshold value is set and the candidate CCR is selected, the user starts the semantic correlator by selecting "Filter Using Keywords." The semantic correlator first retrieves the keyword list for the candidate CCR, which is stored in the *CCRSemantics* component of the FIOM Framework. Then, the semantic correlator iterates through the FIOM and evaluates the keyword list for each FEV FCR in the FIOM. During the evaluation, the percent relevance for each FEV FCR is calculated using Eq 3.7 and saved as the FEV keyword score. The process is then repeated for the next FEV FCR. After all the FEVs have been evaluated, the semantic correlator orders the results according to keyword match score and returns the ordered list of FEVs to the OOMI IDE. The OOMI IDE then displays FEVs whose keyword match score exceeds the threshold value. This enables the interoperability engineer to adjust the threshold value on the returned results and adjust the number of FEVs displayed. Figure III-12 illustrates the process.

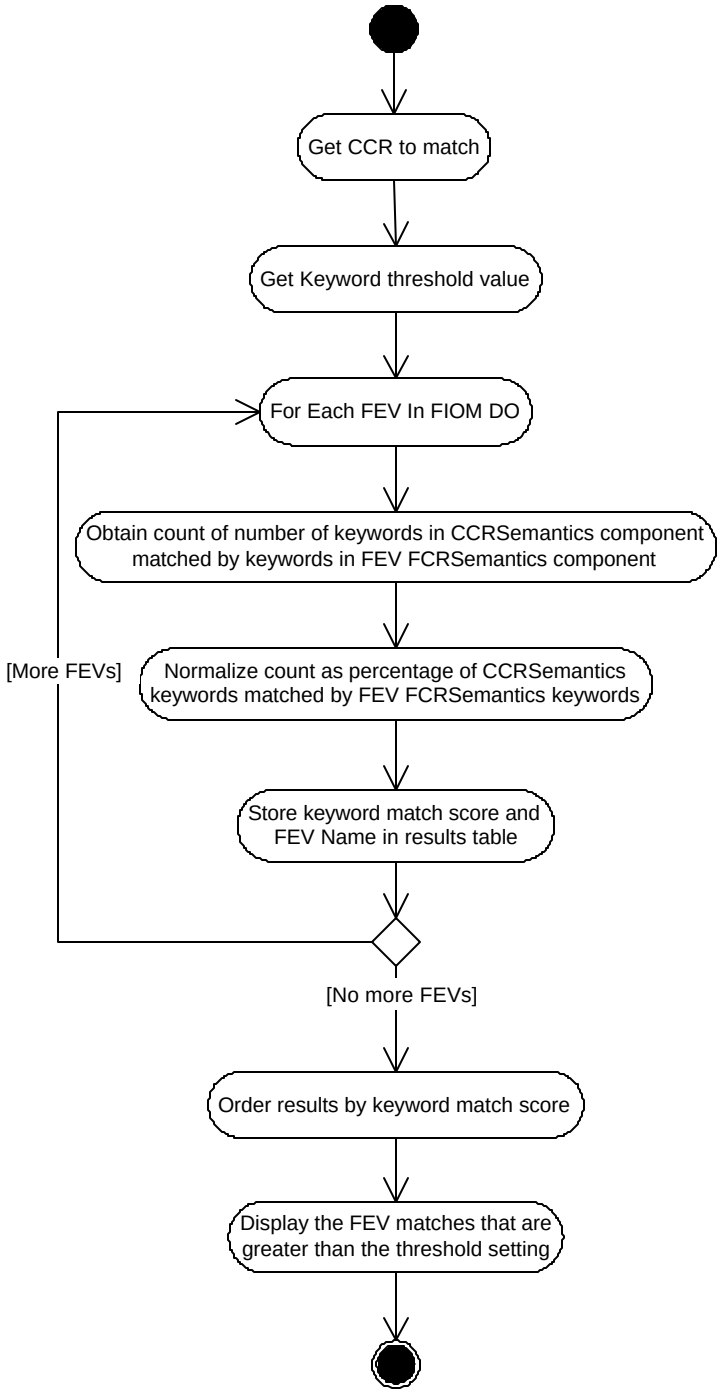


Figure III-12. UML activity diagram of the component model correlator semantic search process.

F. SYNTACTIC CORRELATION

The second phase of the component model correlator is the syntactic search process performed by a syntactic correlator. The syntactic search process, which follows

the semantic search phase, attempts to find a one-to-one correlation between a candidate CCR and a specific FEV FCR through the use of trained neural networks and pattern matching. Before initiating the syntactic correlator the interoperability engineer must perform two steps. First, the threshold setting can be set or the user can accept the default value set by the OOMI IDE. The threshold setting for the syntactic correlator is used in the same fashion as the semantic correlator. The value limits the displayed results in order to decrease the number of FEVs that require final analysis by the interoperability engineer. Second, the interoperability engineer must select the specific FEVs that are to be passed into the syntactic correlator. This is done by manually selecting each FEV in the component model correlator window of the OOMI IDE or selecting the "Select All" option.

After performing the two required steps listed above, the interoperability engineer initiates the syntactic correlator by selecting the "Filter Using Neural Net" option in the correlator window. The syntactic correlator first gets the syntactic components (the discriminator vectors) of the candidate CCR from the *CCRSyntax* component of the FIOM Framework. Next, the syntactic correlator iterates through the list of FEVs performing the syntactic evaluation for each FEV FCR. The evaluation is conducted using the FCR's trained neural network retrieved from the *FCRSyntax* component of the FIOM Framework. The specific details of the evaluation process are discussed in the following paragraphs. A syntactic match score is obtained through each evaluation and is stored with the FEV in the results list. Once all the FEVs have been evaluated, the results list is ordered by syntactic score and returned to the OOMI IDE. The OOMI IDE then displays only those FEVs that score higher than the syntactic threshold setting. Like the semantic search process, this enables the interoperability engineer to adjust the threshold value on the returned results and adjust the number of FEVs displayed. Figure III-13 illustrates the syntactic correlation phase of the component model correlator.

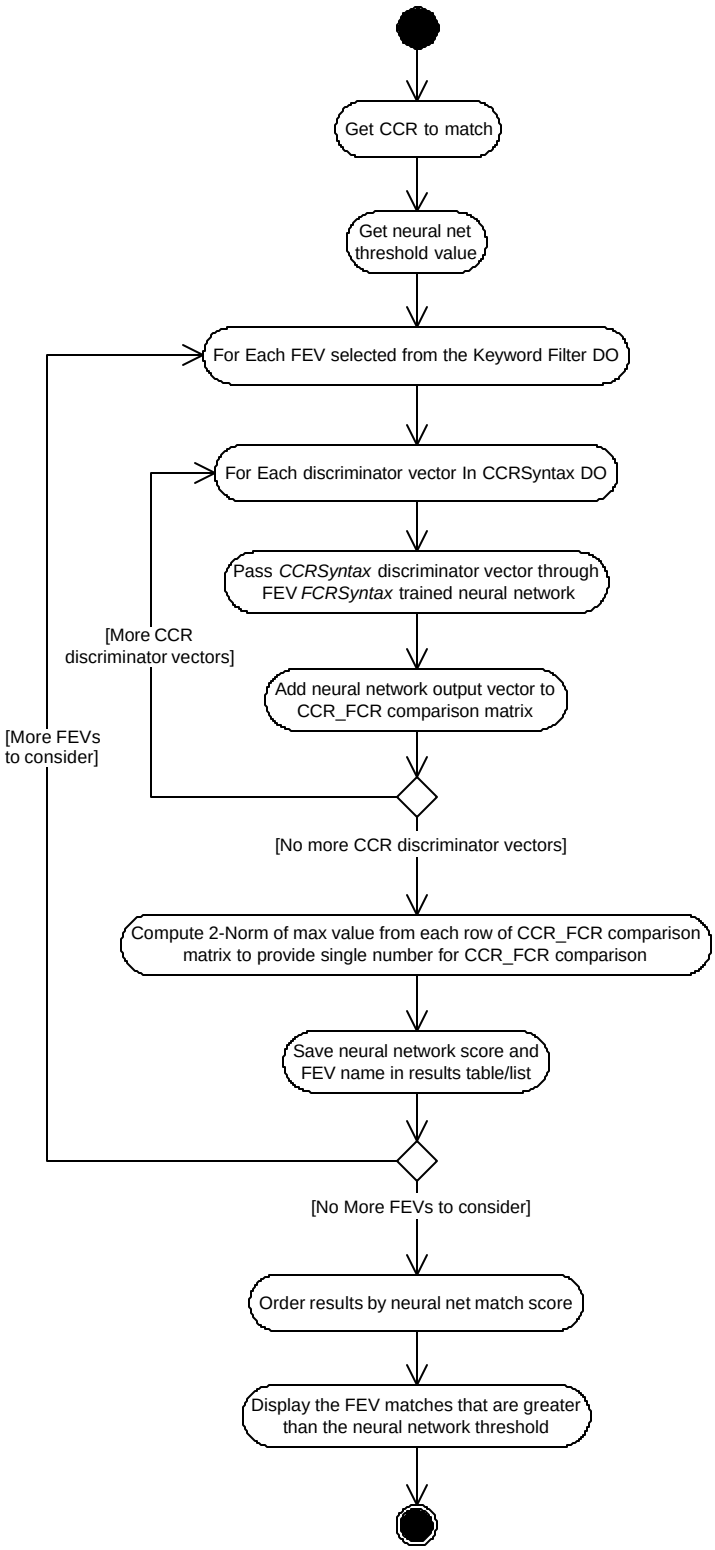


Figure III-13. UML activity diagram of the component model correlator syntactic search process.

During the evaluation of the syntactic components, the discriminator vectors for a candidate CCR are passed through the trained backpropagation neural network of an FEV FCR. A notional example of this process is shown in Figure III-14. In the example, a CCR discriminator vector with the values

$$< 1.0, 0.0, 1.0, 0.321, 0.463 >$$

is passed through the trained neural network. The output vector is calculated to be

$$< 0.00642, 0.89123, 0.23451, 0.03123 >.$$

As discussed in Section III.D.2, each output node of an FEV FCR neural network is used to represent a single FCR attribute. If an input vector correlates to an FCR attribute, the corresponding attribute node in the output layer will output a value close to 1.0, while the remaining nodes will output a value close to 0.0. In this fashion, binary output vectors can be used to facilitate pattern matching. In the example given in Figure III-14, the value from node 2 in the output layer is closer to 1.0 than the remaining output node values. This means that the CCR attribute being evaluated more closely correlates to attribute ß of the FEV FCR than any other attribute. This does not necessarily mean that the attributes are an exact match with one another. Rather, the result indicates that the syntactic structure of the CCR attribute closely correlates to the syntactic structure of the FEV FCR attribute ß.

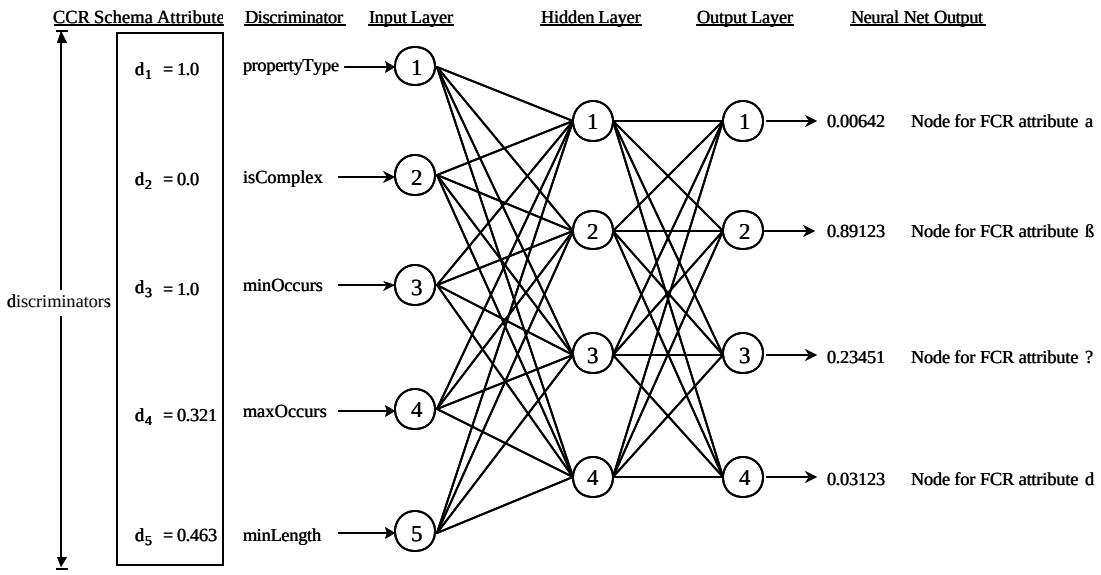


Figure III-14. Notional diagram of a backpropagation neural network showing a forward pass of a CCR schema attribute with the resulting output. [after You02]

In order to establish a correlation score between an entire CCR and FEV FCR, a CCR-FCR correlation matrix is used. [You02] As each candidate CCR attribute is passed through the neural network of an FEV FCR, the output vector is recorded in a row of the comparison matrix. Figure III-15 shows the completed comparison matrix for the armoredFightingVehicle CCR and groundCombatVehicle_View3 FCR comparison.

		FCR Schema Attributes				Row Maximun	
		Type	Position	Time	Status	(X)	(X^2)
CCR Schema Attributes	afvClassification	0.8319	0.0013	0.0002	0.1464	0.8319	0.6921
	afvLocation	0.0056	0.9875	0.0013	6.2e-6	0.9875	0.9752
	afvObsTime	1.6e-7	0.0029	0.9911	0.0374	0.9911	0.9823
	afvStatus	0.0849	0.0003	0.0008	0.9201	0.9201	0.8464
						$\sqrt{\Sigma(X^2)}$	**1.8698**
						(As Percent of Maximum 2-Norm)	**93.49**

Figure III-15. Computing single value for CCR-FCR Comparison Matrix. [after You02]

Once the CCR-FCR comparison matrix is complete, an overall comparison score is obtained by computing the 2-norm of the maximum value from each row. The 2-norm or length of a vector is computed by taking the square root of the sum of the squares of each element in the vector [Ant94]. The ratio of the computed 2-norm of the row maximums to a perfect CCR-FCR match is the final value saved for the FCR comparison. Since the FCR neural networks are trained using binary desired output vectors, the maximum possible value for any row is 1.0. Hence, the theoretical 2-norm for a perfect match is simply

$$\sqrt{num\ FCR\ attributes} \qquad \text{[Eq 3.8]}$$

which becomes the divisor in the ratio calculation. The 2-norm provides a higher score to an FCR, which provides a perfect or near-perfect match for one or more CCR attributes than to an FCR that doesn't provide a close match with any of the CCR attributes. As Figure III-15 shows, the groundCombatVehicle_View3 FCR has four attributes, giving a perfect 2-norm of 2.0. The calculated 2-norm of the comparison with the armoredFightingVehicle CCR is 1.86, resulting in 93.5% score. This value can be interpreted to mean that the groundCombatVehicle_View3 FCR is 93.5% syntactically relevant to the armoredFightingVehicle CCR.

A high syntactic correlation score does not guarantee a one-to-one correspondence between a candidate CCR and an FEV FCR. Using the 2-norm method alone, there are two cases for potential false-positive results. First, there may be cases when multiple attributes or operations of a CCR resemble a single attribute or operation of an FCR. This would result in a false-positive CCR-FCR Comparison Matrix as shown in Figure III-16. In this case, the percentage score obtained by the comparison is identical to the score computed in Figure III-15. However, the CCR attributes *afvLocation* and *afvObsTime* both correlate to the FCR attribute *Time.* Similarly, the CCR attributes *afvClassification* and *afvStatus* both correlate to the FCR attribute *Type*. CCR-FCR correspondence requires a one-to-one correspondence between the attribute and operation sets. Hence, this scenario does not result in correspondence between the CCR and FCR and is considered a false positive. In the second case, the number of attributes of a CCR may not match the

number of attributes of an FCR. If the number of CCR attributes is less than the number of FCR attributes, the CCR-FCR comparison matrix should contain missing column values since one-to-one correspondence is impossible. Similarly, if the number of CCR attributes is greater than the number of FCR attributes, the CCR-FCR comparison matrix should contain duplicate column matches. However, in both situations, the 2-norm score may still be high and produce a false-positive result.

		FCR Schema Attributes				Row Maximun	
		Type	Position	Time	Status	(X)	(X^2)
CCR Schema Attributes	afvClassification	0.8319	0.0013	0.0002	0.1464	0.8319	0.6921
	afvLocation	0.0056	0.0013	0.9875	6.2e-6	0.9875	0.9752
	afvObsTime	1.6e-7	0.0029	0.9911	0.0374	0.9911	0.9823
	afvStatus	0.9201	0.0003	0.0008	0.0849	0.9201	0.8464
						$\sqrt{\Sigma(x^2)}$	**1.8698**
						(As Percent of Maximum 2-Norm)	**93.49**

Figure III-16. Example of a false-positive syntactic correlation.

A method to avoid false positives in the syntactic correlation process involves two steps. First, the number of attributes of a CCR is compared to the number of attributes of an FCR. Only FCRs with an equal number of attributes are syntactically evaluated. Second, a record of unique attribute matches is maintained during the evaluation of an FCR. In this step, an output vector produced from an FCR's neural network is checked to identify which attribute it most closely correlates with. The check is performed by determining the attribute position of each row maximum in the comparison matrix. A record can then be maintained to indicate whether or not an FCR attribute has received a potential match or not. Once the comparison matrix is complete, the number of matches can then be used as a true or false filter for the FCRs with a comparison matrix score above the threshold setting. For example, consider the comparison matrix in Figure III-15. As the matrix is constructed, a record indicates that all 4 FCR attributes have been potentially matched. The record may be in the form of a boolean array sized to match the number of FCR attributes. A true value in the array indicates that the corresponding

attribute or operation has been matched at least once. A false indicates that the comparison matrix includes no matches for that attribute or operation. Since the number of matches for a one-to-one correspondence must exactly equal the number of attributes or operations defined in the FCR, one or more false values in the results record eliminates the FCR as a potential match to the CCR despite a high 2-norm score. Using this method, the comparison matrix shown in Figure III-16 produces a count of 2 attribute matches in the results array instead of 4. Thus, the potential false positive is eliminated from consideration.

The method described above is valid since CCR-FCR correspondence requires a one-to-one correspondence between the attribute and operation sets. If an FCR has n attributes or operations, then one-to-one correspondence requires n matching attributes or operations in the CCR. Therefore, any match count less than the number of FCR attributes does not have one-to-one correspondence. In other words, any FCR attribute or operation with a match record set to 0 or false, would fail the comparison matrix. However, since there are cases when the interoperability engineer may want to analyze all results, the one-to-one correspondence option can be turned off as an option in the OOMI IDE.

In order to use this method to eliminate false-positive results, an attribute threshold value must be used to determine whether or not a row maximum is high enough to infer correlation between a CCR input vector and the FCR attribute. Since the FCR neural network uses binary desired output vectors, a CCR attribute or operation correlates to an FCR attribute or operation when one of its values is close to 1.0 and the remaining values are close to 0.0. However, the values can be in the range [0.0,1.0] as illustrated in Figure III-15. The row maximums that indicate correspondence are close to the upper bound of the potential range, but a distinct threshold value is needed in order to conclude that the row maximum indicates a potential match. The threshold can be the same value as the neural network threshold setting in the correlator panel, or it can be an entirely different threshold setting. The interoperability engineer must determine the exact value.

G. SUMMARY

The OOMI IDE component model correlator methodology is used to assist the interoperability engineer in adding a Component Class Representation (CCR) to a Federation Interoperability Object Model (FIOM) during the *Register Component Class Representation* phase of the overall FIOM construction process within the OOMI IDE. The goal of component model correlation is to assist the interoperability engineer in the construction of translations between component systems and the federated ontology by identifying potential matches in data models between a component system and the FIOM.

The component model correlator uses a two-phase approach to establish correspondence between component models and federation models. In phase one, a semantic, or keyword matching process takes place using keyword information extracted from specific elements of a CCR and FCR. In the second phase, details about the structure and composition of the attributes and operations used to model the real-world entity are used in conjunction with neural networks to provide syntactic correlation. Each phase of the correlation provides a score to the interoperability engineer. Based on the scores of the semantic search, the interoperability engineer can select a set of potential FEV matches to be passed to the second phase. Comparisons of the overall scores among potential FCR matches guide the interoperability engineer toward the most likely match for a CCR. However, final determination of CCR-FCR correspondence requires a one-to-one correspondence between the attribute and operation sets of a potential CCR-FCR match. This final determination of equivalence is the responsibility of the interoperability engineer and is not automated in the OOMI IDE.

IV. COMPONENT MODEL CORRELATOR IMPLEMENTATION

A. COMPONENT MODEL CORRELATOR MODULE

As discussed in Section III.B, the Component Model Correlator is a module within the OOMI IDE that is responsible for establishing correspondences between component and federation models of a real-world entity in the FIOM. Figure III-2 illustrates the position of the component model correlator module in relation to the other major components of the OOMI IDE. The Component Model Correlator module implements the methodology described in Chapter III. It generates the semantic components required for the semantic correlation phase and generates the necessary syntactic components for the syntactic correlation phase. Additionally, the module performs both semantic and syntactic correlation of a component system CCR to an FEV in the FIOM.

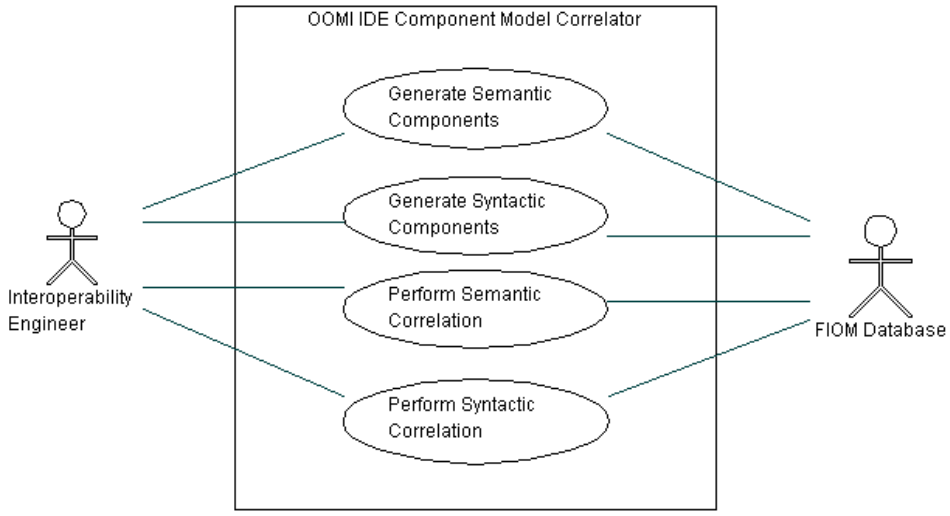


Figure IV-1. UML Use Case diagram for Component Model Correlator.

The use case diagram developed for the implementation of the Component Model Correlator is shown in Figure IV-1. The two primary actors involved in these use cases are the interoperability engineer and the FIOM Database. The interoperability engineer interacts with the Component Model Correlator through the user interface of the OOMI IDE. Each of the four use cases identified interacts with FIOM Database, which is the

repository for the FIOM components. The use cases above directly correspond to the aspects of the correlator methodology discussed in Chapter III.

The abstract architecture shown in Figure IV-2 converts the Component Model Correlator use cases into modular subcomponents. The *ComponentModelCorrelator* subcomponent is the executive of the entire module. It receives commands from the OOMI IDE then initiates the appropriate action with either a *Generators* subcomponent or a *Correlators* subcomponent. The *Generators* subcomponent is an abstraction of both a *Semantic* and *Syntactic Component Generator*. The *Semantic* and *Syntactic Component Generators* implement the semantic and syntactic component generation methodology discussed in Section III.C and Section III.D respectively. Similarly, the *Correlators* subcomponent is an abstraction of both a *Semantic* and *Syntactic Correlator*. The *Semantic* and *Syntactic Correlators* implement the semantic and syntactic correlation methodology outlined in Section III.E and Section III.F respectively.

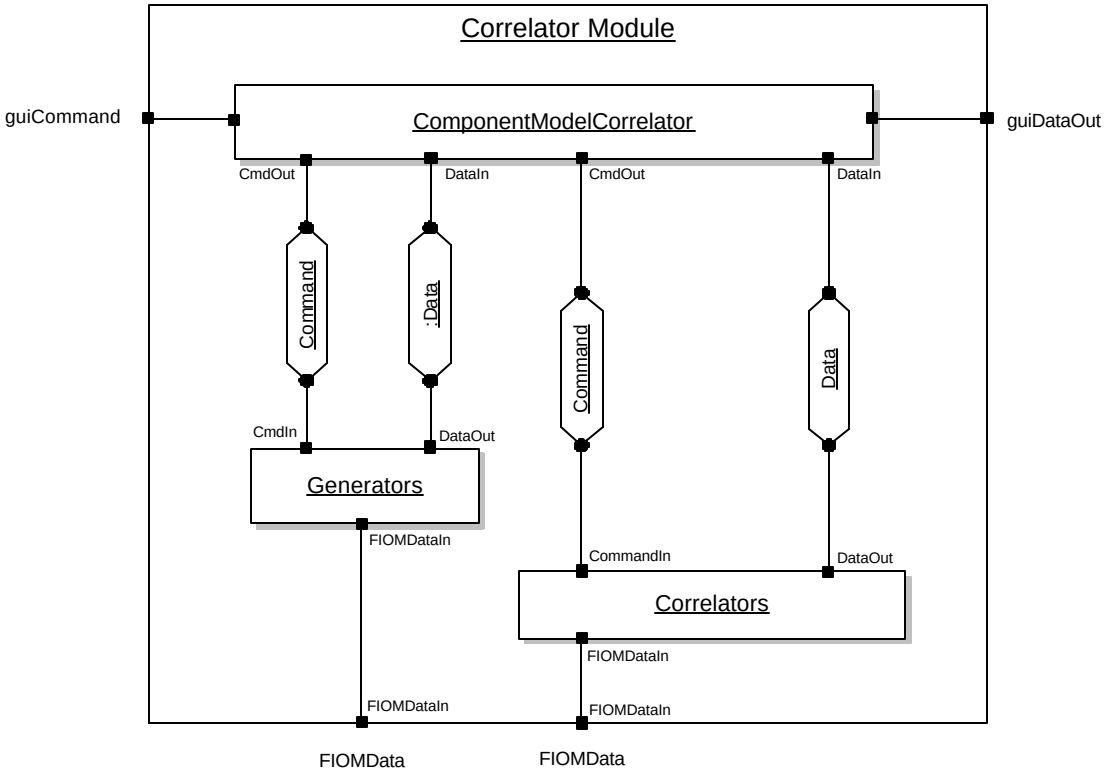


Figure IV-2. Abstract Component Model Correlator architecture.

After receiving a command from the *ComponentModelCorrelator* executive, the *Generators* or *Correlators* subcomponent completes the required action and passes data back to the *ComponentModelCorrelator*. The *ComponentModelCorrelator* then performs any data formatting or data conversion required prior to sending that output data back to the OOMI IDE. This architecture adheres to standard software engineering principles, particularly abstraction, modularity and the principles of cohesion and coupling. The architecture is both scalable and extendable. This is very important since the methodologies for the *Generators* and *Correlators* subcomponents will undoubtedly be revised or completely changed over time.

This thesis implements a prototype of the architecture presented in Figure IV-2. The prototype Component Model Correlator module is coded in Java using Sun's JDK 1.3.1_04. The classes and packages of the module are shown in Figure IV-3 along with important associations with other packages of the OOMI IDE prototype. As shown, the modular architecture converts easily into the Java package framework. The *Generators* and *Correlator*s architectural subcomponents map to respective semantic and syntactic generator packages and semantic and syntactic correlation packages in the Java implementation. However, the *ComponentModelCorrelator* subcomponent is not a separate package. Rather, the component is realized in the *ComponentModelCorrelator* class directly under the *Correlator* Java package. The Java implementation shown in Figure IV-3 makes use of Java interfaces at all levels. The interfaces define a contract of services between the *ComponentModelCorrelator* executive and the implementations of the subcomponents. A brief description of each item shown in Figure IV-3 follows the figure. A complete source code listing of the prototype Component Model Correlator module is included in Appendix A.

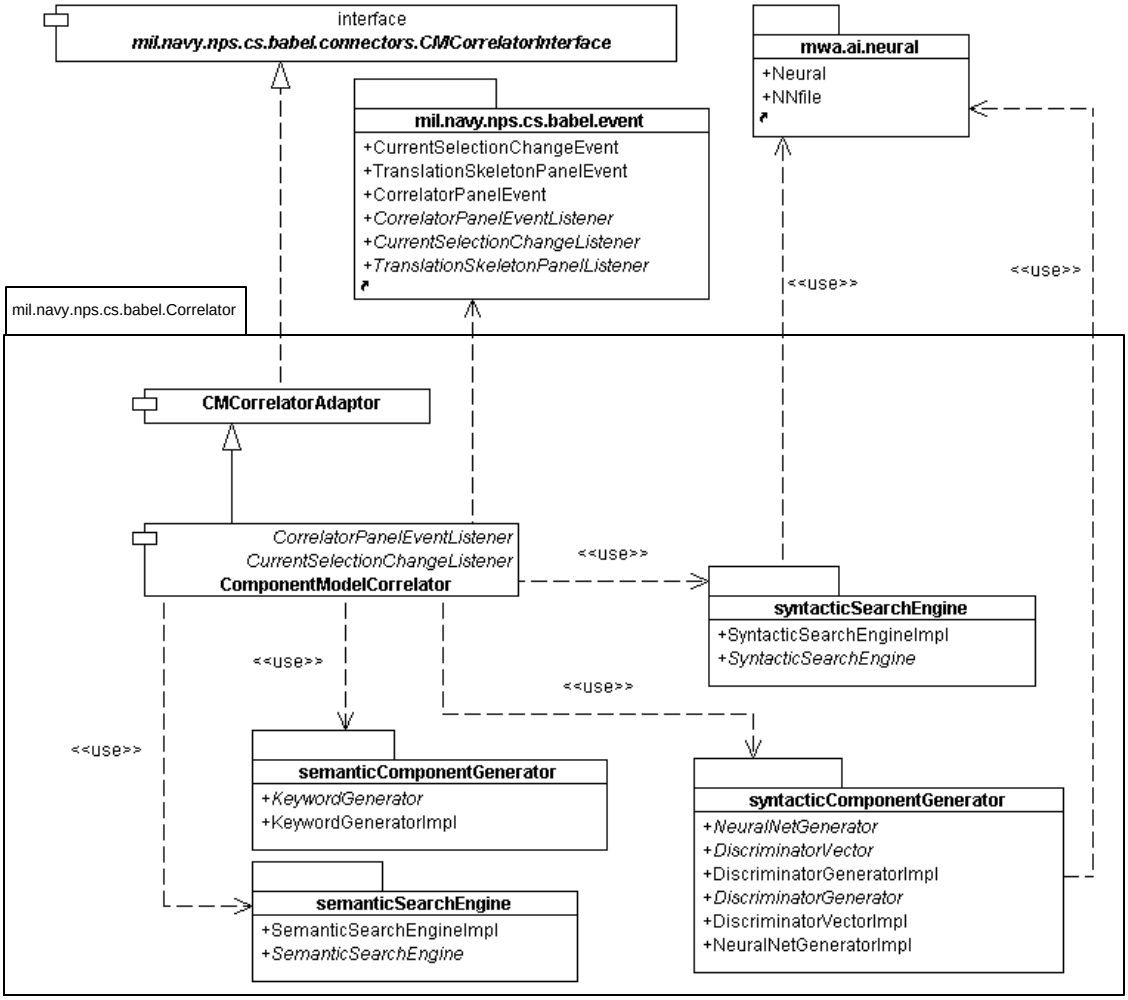


Figure IV-3. UML diagram showing the packages of the Component Model Correlator and the classes contained in each package.

mil.navy.nps.cs.babel.connectors – An OOMI IDE package that contains the interfaces for all subcomponents of the OOMI IDE. The interfaces prevent high coupling between components and the OOMI IDE by establishing well-defined services.

CMCorrelatorInterface – An interface that defines the services required by the Component Model Correlator Module. The methods associate directly to the use cases identified in Figure IV-1.

mil.navy.nps.cs.babel.event – An OOMI IDE package that contains the event and event listener classes for all components of the OOMI IDE. Events and event listeners are

used in user interfaces as alternative execution paths to components rather than using explicit method calls through the subcomponents interface. An event is characterized as an interaction action between a user and the user interface. Examples of events include mouse clicks, mouse movement, and keyboard actions. A GUI control, such as a button or menu can be told to "listen" for particular events. When the event occurs, any class registered with the appropriate event listener act upon the event.

CorrelatorPanelEvent – A class that defines the events associated with the Correlator panel. The specific events are discussed in Section IV.C.1.

CorrelatorPanelEventListener – An interface that is implemented by the *ComponentModelCorrelator* class. It enables the Correlator to listen for events that take place within the Correlator Panel.

mil.navy.nps.cs.babel.Correlator – A package containing the classes and interfaces of the Component Model Correlator module.

CMCorrelatorAdaptor – A class that implements the *CMCorrelatorInterface* with methods that throw a "method not implemented" exception. The *ComponentModelCorrelator* class is a generalization of this class.

ComponentModelCorrelator – The overall executive of the Component Model Correlator model. This class coordinates the use of the services contained in the sub-packages and handles the duties of sending information back to the OOMI IDE. It inherits from *CMCorrelatorAdaptor* and overrides the methods of the super class for which an implementation exists. Otherwise, *CMCorrelatorAdaptor* will throw a "method not implemented" exception.

mil.navy.nps.cs.babel.Correlator.semanticComponentGenerator – The package containing the classes used to generate the semantic components required for the semantic search phase of the Component Model Correlator.

KeywordGenerator – The interface defining the services for a keyword generator. A keyword generator creates a keyword list for a CCR and an FCR. The keyword list is

the semantic component required for the semantic correlation phase. The methodology for semantic component generation is discussed in Section III.C.

KeywordGeneratorImpl – An implementation of a keyword generator, which generates the semantic components of a CCR and an FCR. This class implements the semantic component generation methodology outlined in Section III.C.

mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator – The package containing the classes used to generate the syntactic components required for the syntactic search phase of the Component Model Correlator.

DiscriminatorGenerator – The interface defining the services required for a discriminator vector generator as outlined in the syntactic component generation methodology discussed in Section III.D.1. A discriminator generator generates the discriminator vectors for a CCR or an FCR required for the syntactic correlation phase.

DiscriminatorGeneratorImpl – An implementation of a discriminator vector generator, which generates the discriminator vectors for a CCR and an FCR. This class implements the methodology discussed in Section III.D.1.

NeuralNetGenerator – The interface defining the services for a neural network generator. The neural network generator creates and trains a neural network for an FCR. The neural network is an additional syntactic component of an FCR used in the syntactic correlation phase as discussed in Section III.D.2.

NeuralNetGeneratorImpl – An implementation of a neural network generator that creates and trains an FCR neural network. This class implements the methodology discussed in Section III.D.2.

DiscriminatorVector – The interface defining the services for a discriminator vector utility class. This interface defines the property get/set methods of a discriminator vector as defined in Section III.D.1.

DiscriminatorVectorImpl – An implementation of a discriminator vector utility class. This class contains the logic used for the creation of a discriminator vector as well

as the calculations required to normalize a vector. This class implements the methodology discussed in Section III.D.1.

mil.navy.nps.cs.babel.Correlator.semanticSearchEngine – The package containing the component classes that implement the semantic search phase of Component Model Correlator.

SemanticSearchEngine – The interface defining the services required for a semantic search engine. The semantic search engine performs the semantic correlation as discussion in Section III.E.

SemanticSearchEngineImpl – An implementation of a semantic search engine that performs the semantic correlation phase of the Component Model Correlator. This class implements the methodology discussed in Section III.E.

mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine – The package containing the component classes that implement the semantic search phase of Component Model Correlator.

SyntacticSearchEngine – The interface defining the services required for a syntactic search engine. The syntactic search engine performs the syntactic correlation as discussion in Section III.F

SyntacticSearchEngineImpl – An implementation of a syntactic search engine that performs the syntactic correlation phase of the Component Model Correlator. This class implements the methodology discussed in Section III.F.

mwa.ai.neural – The package from [Wat97] containing the neural network component classes used within the Component Model Correlator.

Neural – An implementation of a backpropagation neural network adapted from [Wat97]. This class contains all methods for creating, training and using an instance of a neural network.

NNFile – A utility input-output class that parses a Neural object to and from a specifically formatted file.

As defined in the architecture of Figure IV-2, the four subcomponents of the Component Model Correlator, *semanticSearchEngine*, *syntacticSearchEngine*, *semanticComponentGenerator*, and *syntacticComponentGenerator*, are cohesive modules that decouple the implementation of these features from the user interface (UI) components of the OOMI IDE. The link between the Component Model Correlator module features and the OOMI IDE UI is the *ComponentModelCorrelator*. As stated, this class is the executive of the Correlator module and is coupled to the OOMI IDE UI by the *CMCorrelatorInterface* and the *CorrelatorPanelEventListener*.

1. Classes in package mil.navy.nps.cs.oomi.babel.Correlator

The classes contained in the Correlator package and their associations with other components of the OOMI IDE are shown in Figure IV-4. These classes provide the linkage between the OOMI IDE UI implementation and the UI-independent services of the Component Model Correlator. The *CMCorrelatorInterface* in package mil.navy.nps.cs.oomi.babel.connector defines the core services the OOMI IDE expects from the *ComponentModelCorrelator*.

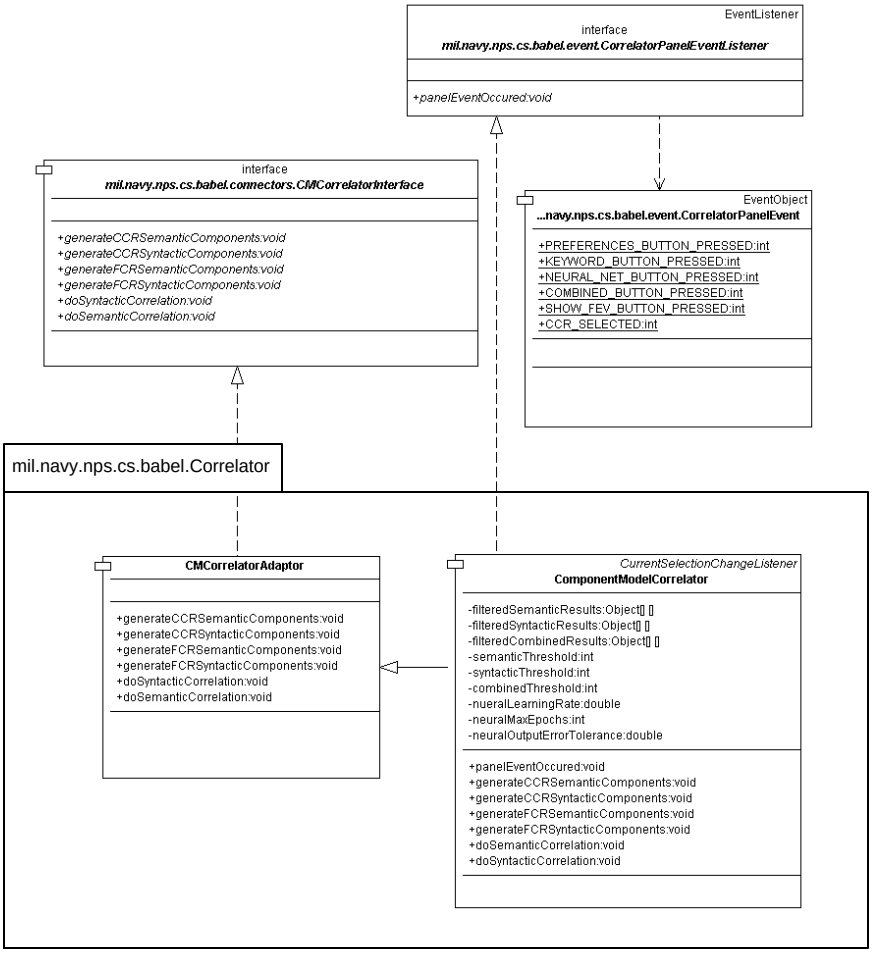


Figure IV-4. Class diagram of the Correlator package showing the important class methods.

The *CMCorrelatorAdaptor* class is designed to assist in the software development process. It implements the *CMCorrelatorInterface* interface but throws the exception *java.lang.UnsupportedOperationException* for every operation defined in the interface. The OOMI IDE handles the exception by displaying an alert message stating that the operation being called has not yet been implemented. The *ComponentModelCorrelator* then extends the *CMCorrelatorAdaptor* and overrides only those operations with implementations. Using this technique, the features of the OOMI IDE UI provide feedback to the developers despite the fact that implementations may not exist yet.

The implementation of the Component Model Correlator presented in this thesis allows the OOMI IDE to access the module features using two distinct paths. During the

Add Component System External Interface and *Manage Federation Entities* phases of the OOMI IDE, the features of the Correlator module are accessed through the *CMCorrelatorInterface*. However, during *Register Component Class Representation* phase, event listeners call the Correlator features. The *ComponentModelCorrelator* implements the *CorrelatorPanelEventListener* interface as shown in Figure IV-4. The class *CorrelatorPanelEvent* defines the common events associated with the component model correlator panel within the OOMI IDE. When a feature on the correlator panel is executed it fires a *CorrelatorPanelEvent* instance, which is then acted upon by the *ComponentModelCorrelator*. A more detailed discussion of the *CorrelatorPanel* user interface is provided in Section IV.C.1

2. Modifications to the FIOM Framework

The implementation of the Component Model Correlator required additions and modifications to the Federation Interoperability Object Model (FIOM) framework described in Section II.D.4 and originally developed by Lee [Lee02]. As shown in Figure IV-5, The FIOM framework is contained within the package mil.navy.nps.cs.oomi.fiom.

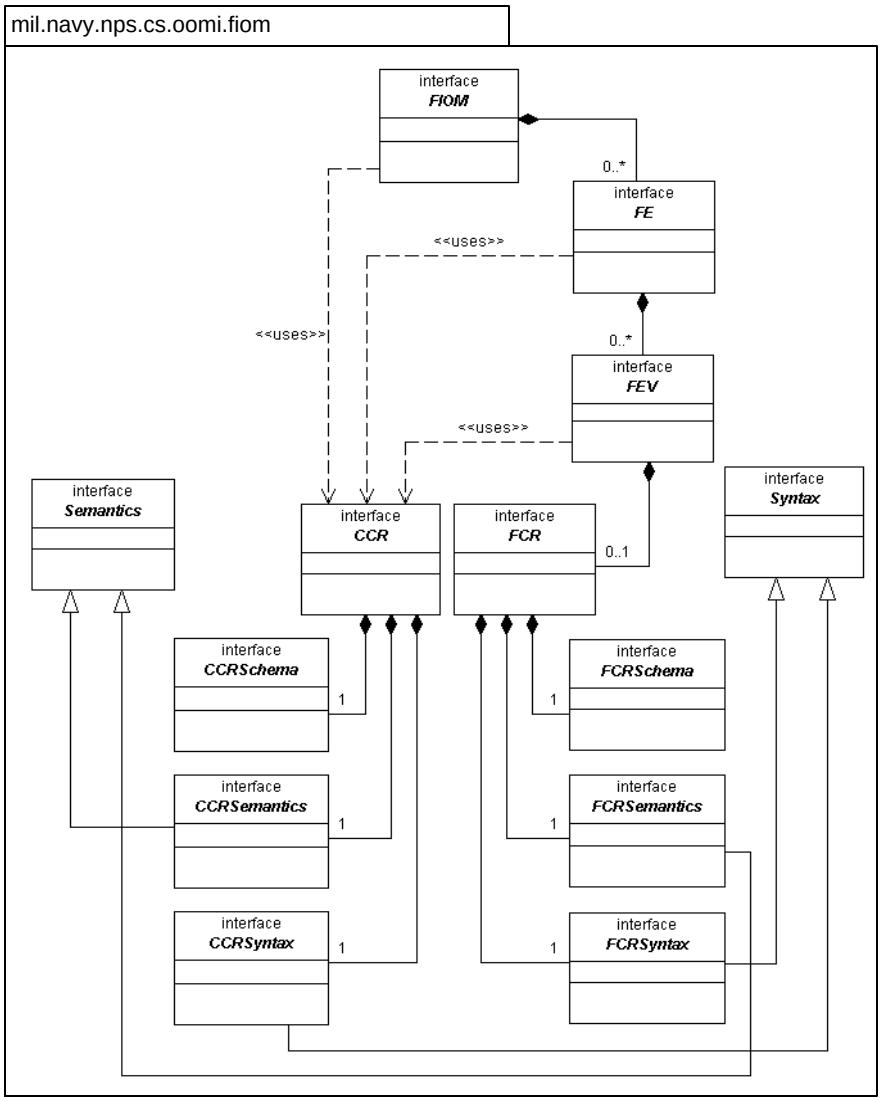


Figure IV-5. Additions and modifications to the FIOM framework.

The original FIOM framework did not provide an implementation for the *CCRSemantics*, *FCRSemantics*, *CCRSyntax*, and *FCRSyntax* components. These components are required for the Component Model Correlator and have been implemented in this thesis. Additionally, the original *CCR* and *FCR* components did not include the composition associations for the *CCRSemantics*, *FCRSemantics*, *CCRSyntax*, and *FCRSyntax* components as indicated in Figure IV-5. As such, the composition associations have been included in the *CCR* and *FCR* components as well as a number of property get/set methods. Lastly, *Semantics* and *Syntax* components were added, to define

the services required by the *CCRSemantics*, *FCRSemantics*, *CCRSyntax*, and *FCRSyntax* components. The complete source code listing of the FIOM Framework additions and modifications is included in Appendix B.

B. COMPONENT GENERATION

1. XML IMPLEMENTATION Considerations

As stated in Section III.C and III.D, semantic and syntactic components for the Component Model Correlator are generated from a CCR XML Schema file or an FCR XML Schema file. Three XML processing APIs were evaluated for use in the implementation of the component model correlator.

DOM The Document Object Model (DOM) is a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The DOM specification is a recommendation from the World Wide Web Consortium (W3C) and was first entitled DOM Level 1. The current version is DOM Level 3. [HH02] The DOM can be used with many types of documents and provides a means for working with documents in code. In terms of XML, developers can use the DOM to create XML documents or parts of XML documents, navigate through a document, and move, copy or delete parts of an XML document. As inferred from the name, the DOM represents the entire document, an important concept. This method enables fairly easy processing of an XML document. However, DOM requires a large amount of memory to represent an XML document, which can be critical in small footprint devices. [HCD+01] A big plus for DOM is that it takes a generic approach to modeling XML documents regardless of structure. The DOM is usually added as a layer between an XML parser and the application using the XML document as shown in Figure IV-6. An XML parser processes an XML document into the DOM representation. The application can then get information about the XML document using the DOM interfaces.

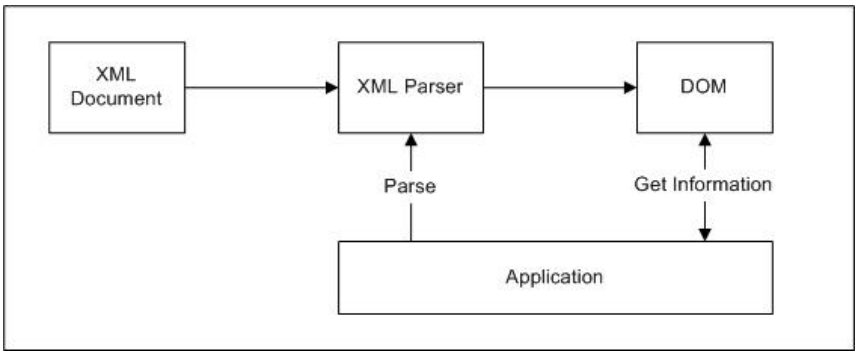


Figure IV-6. The DOM approach to obtaining information from an XML document. [HCD+01]

SAX The Simple API for XML (SAX) is another approach to obtaining information about an XML document. SAX was developed in order to enable more efficient analysis of large XML documents than DOM can provide. As stated above, the DOM approach addresses the entire document. Before an application can use DOM to traverse an XML document, DOM builds up a massive in-memory map of the document that takes space and time. [HCD+01] The DOM approach is extremely inefficient if an application is attempting to extract a small amount of information. The SAX approach is event-driven. Rather than parse an XML document into DOM and then extract information, SAX instructs the parser to raise events when it finds bits of information desired, usually a specific XML tag. The SAX approach is shown in Figure IV-7.

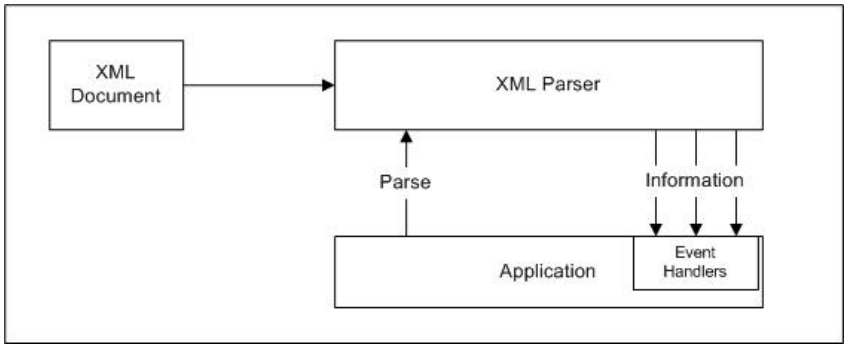


Figure IV-7. The SAX approach to obtaining information from an XML document. [HCD+01]

The interesting aspect of SAX is that no one owns it. It is not a formal recommendation like DOM and it does not belong to any consortium, standards body or

company. It was developed by a group of developers who were looking for a solution to incompatibilities between XML parsers. It survives because it is simple and it works. [HDC+01] Examples of events in SAX 2.0, the current version of SAX, include *startElement*, *endElement*, *startDocument* and *endDocument*. The use of SAX does not eliminate XML parsing. It merely prevents the large overhead associated with DOM and is appropriate when small bits of information need to be extracted from an XML document.

JDOM JDOM provides a Java representation of an XML document for easy and efficient reading, manipulation, and writing. It has a straightforward API, is a lightweight and fast, and is optimized for the Java programmer. It's an alternative to DOM and SAX, although it integrates well with both DOM and SAX. JDOM is not a wrapper for the W3C's DOM. JDOM serves the same purpose as DOM, but is easier to use for Java programmers. Like DOM, JDOM is not an XML parser, it is a document object model that uses XML parsers to build documents. The default XML parser used by JDOM is the JAXP-selected parser from Sun Microsystems. However, JDOM can use nearly any parser. [HM02]

JDOM was chosen for use in the generation of the syntactic and semantics components of the Component Model Correlator. The OOMI IDE is responsible for parsing an XML document into the JDOM representation. The input to the Syntactic and Semantic Component Generators of the Component Model Correlator is a JDOM instance of the XML Schema. The component generation then uses the JDOM interfaces to traverse the document and extract the required information. A more detailed discussion of this process is given in Section IV.B.1 for the Semantic Component Generator and in Section IV.B.3 for the Syntactic Component Generator.

2. Semantic Component Generator

The *mil.navy.nps.cs.babel.Correlator.semanticComponentGenerator* package contains the classes used to generate the semantic components required for the semantic correlation phase of the Component Model Correlator. The ComponentModelCorrelator executive uses these classes when semantic component generation is requested from the OOMI IDE. As shown in Figure IV-8, the *semanticComponentGenerator* contains the

KeywordGenerator interface, which defines the services for a keyword generator, and the implementation class *KeywordGeneratorImpl.*.

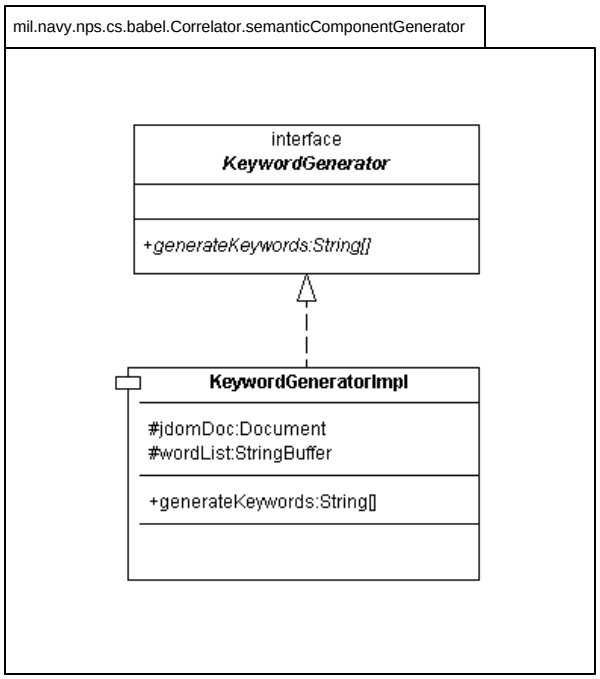


Figure IV-8. UML Class diagram of package *semanticComponentGenerator.*

The generation of the semantic components for a CCR is conducted during the *ADD Component System External Interface* phase of FIOM construction. This process begins when the interoperability engineer (IE) clicks the "Generate Semantics" button as shown in Figure IV-9.

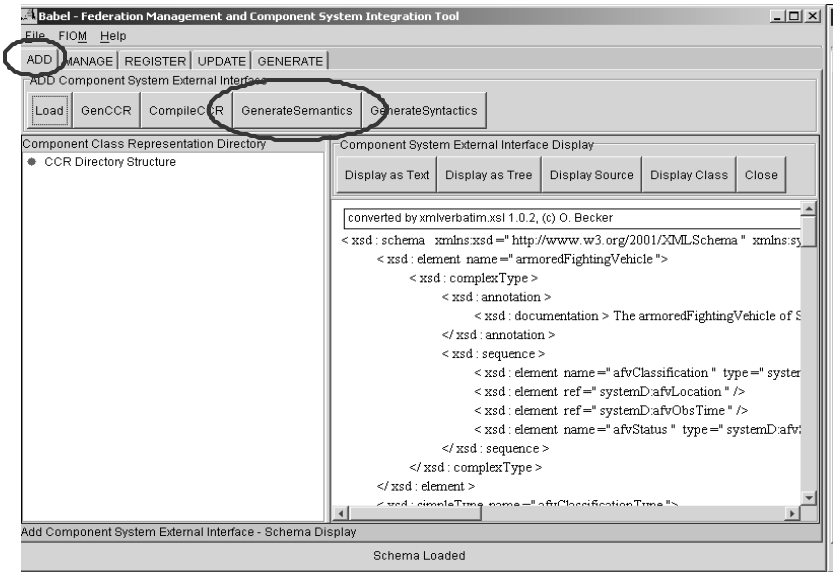


Figure IV-9. Initiating Semantic Component Generation for a CCR.

As depicted in Figure IV-10, when the button is clicked, the OOMI IDE calls the *generateCCRSemanticComponents* method of the interface *CMCorrelatorInterface*, which is implemented by the *ComponentModelCorrelator* class. The *ComponentModelCorrelator* then creates a *KeywordGenerator* instance. The meat of the keyword generation is contained within the *generateKeywords* method of the *KeywordGenerator*. This method takes a JDOM document representing the CCR or FCR as its only parameter and returns an array of Strings holding the values of the keywords. To extract the keywords from the JDOM document, *generateKeywords* calls the private method *parseNode*. This method is a recursive function that takes a JDOM element as its only parameter. The function traverses the JDOM document and analyzes each node encountered in accordance with the Table III-1 of Section III.C. Once the *KeywordGenerator* returns the string of keywords, the *ComponentModelCorrelator* then binds the String array to the FCR or CCR semantic component. The process for generating an FCR keyword list is identical, except that it is initiated during the *Manage Federation Entities* phase of the OOMI IDE.

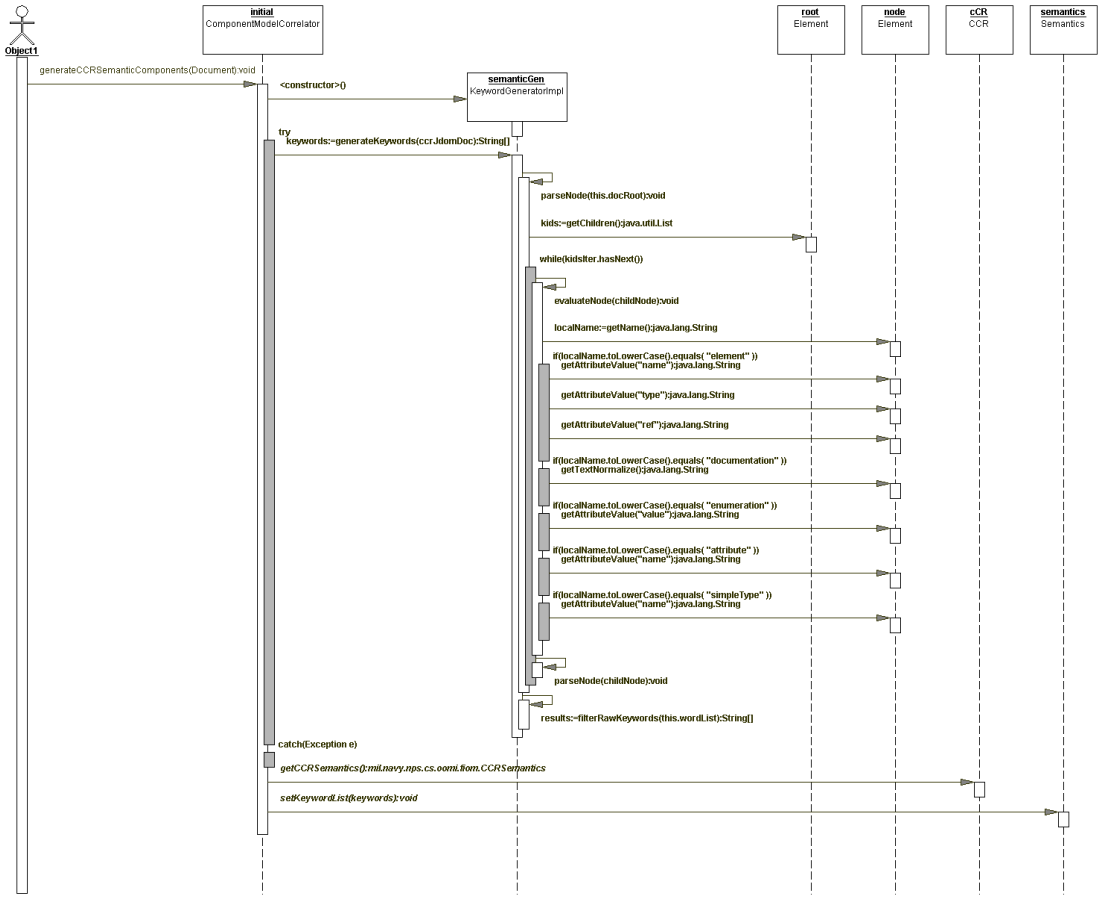


Figure IV-10. UML Sequence diagram of semantic component generation.

3. Syntactic Component Generator

The *mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator* package contains the classes used to generate the syntactic components required for the syntactic correlation phase of the Component Model Correlator. These components include the discriminator vectors for an FCR or CCR and the neural network for an FCR. Figure IV-11 shows the interfaces and classes of the *syntacticComponentGenerator* package. The *ComponentModelCorrelator* manages the generation of the syntactic components through the *NeuralNetGenerator* and *DiscriminatorGenerator* interfaces. Theses two interfaces define the primary methods needed to generate the syntactic components. The *DiscriminatorVector* interface and the *DiscriminatorVectorImpl* class encapsulate the data structure and logic for a single discriminator vector. The DiscriminatorVector interface

also defines two extremely valuable methods, *ValuesAsFloatArray()* and *ValuesAsDoubleArray().* These two methods are used to convert an instance of *DiscriminatorVector* into a format needed by the neural network.

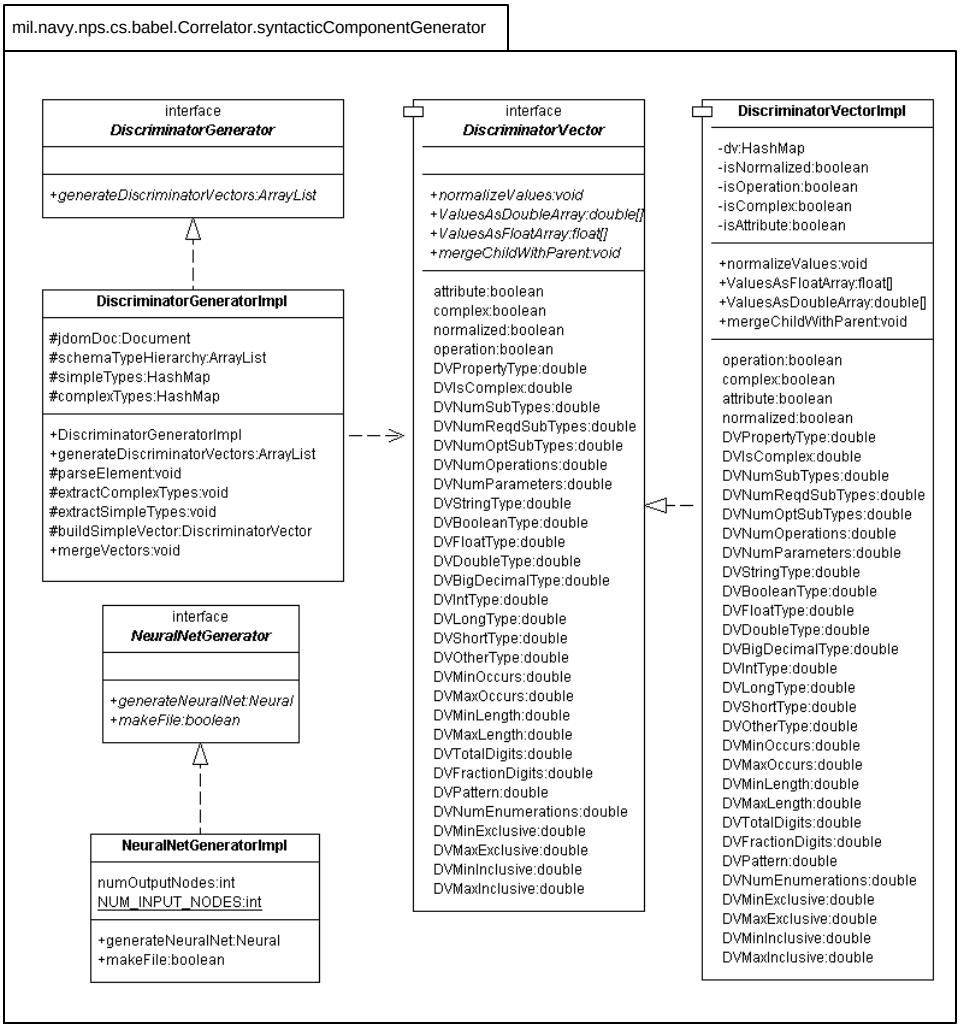


Figure IV-11. UML Class diagram of package *syntacticComponentGenerator*.

The generation of the syntactic components for a CCR is conducted during the *ADD Component System External Interface* phase of FIOM construction. The interoperability engineer initiates the syntactic generation process by clicking the "Generate Syntactics" from "Add" tab in the IDE as shown in Figure IV-12.

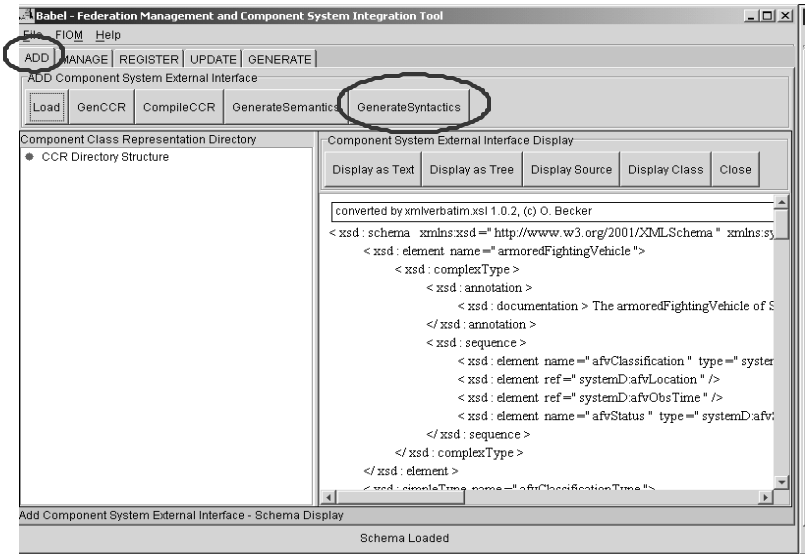


Figure IV-12. Initiating Syntactic Component Generation for a CCR

Of note, the generation of the syntactic components can occur before or after the generation of the semantic components. When the "Generate Syntactics" button is clicked, the OOMI IDE uses the *CMCorrelatorInterface* to access the *ComponentModelCorrelator* method *generateCCRSyntacticComponents*, which takes a JDOM document of the CCR XML Schema as a parameter. The *ComponentModelCorrelator* then creates an instance of *DiscriminatorGeneratorImpl* and calls the *generateDiscriminatorVectors* method. This method traverses through the JDOM document and builds discriminator vectors in accordance with the discriminator vector schema defined in Section III.D.1. The method then returns a Java List that contains an individual float array for each attribute of the CCR. Each float array contains the normalized values of an attribute discriminator vector. The list of discriminator vectors is then associated with the *CCRSyntax* component for use in the syntactic correlation process. Figure IV-13 illustrates the process.

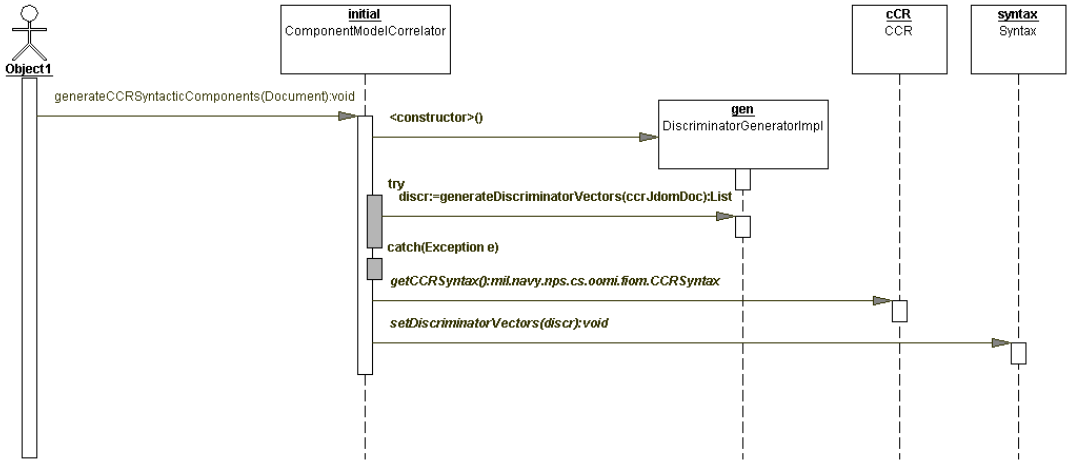


Figure IV-13. UML Sequence diagram showing syntactic component generation for a CCR.

The generation of the syntactic components for an FCR is conducted during *Manage Federation Entities* phase of the OOMI IDE and follows the same initial steps as syntactic component generation for a CCR. Once the discriminator vectors for an FCR have been generated, the *ComponentModelCorrelator* creates an instance of *NeuralNetGeneratorImpl* and calls the *generateNueralNet* method. This method takes a list of discriminator vectors as its only parameter and returns a trained, backpropagation neural network. The neural network is then saved as a file using the *NeuralNetGenerator* method *makeFile*. The *makeFile* method takes a trained neural network and an absolute file path as its parameters and returns a boolean if the file has been created successfully. The neural network file path and the instance of the neural network are then associated with the *FCRSyntax* component for use in the syntactic correlation process. Figure IV-14 illustrates the process for an FCR.

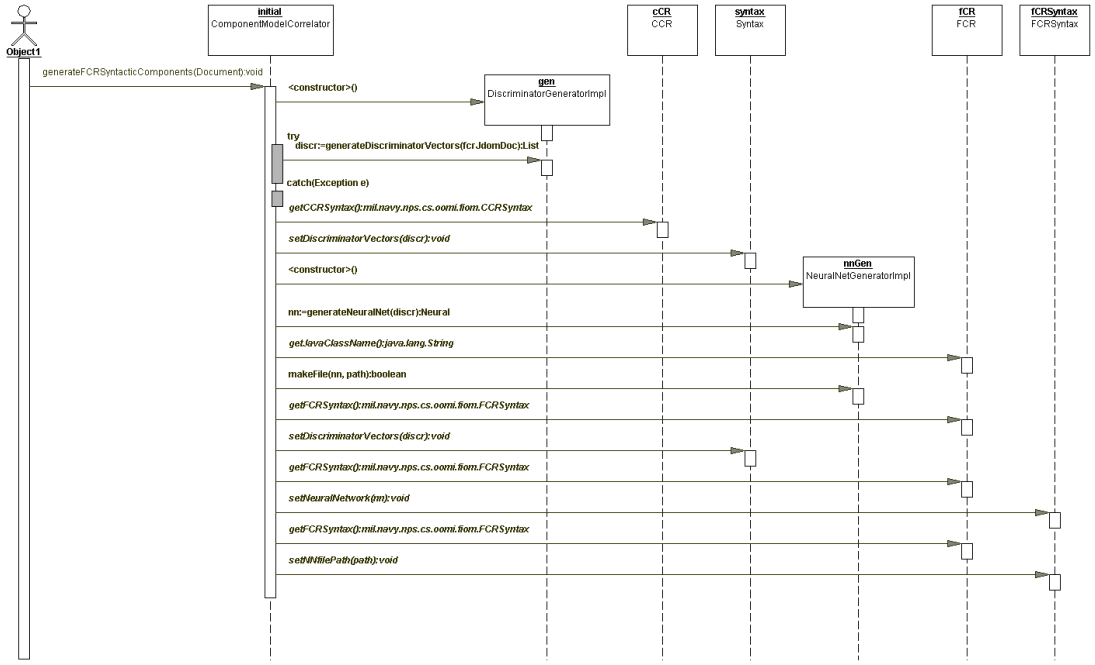


Figure IV-14. UML Sequence diagram showing syntactic component generation for an FCR.

C. CORRELATION

1. Correlator Panel

Component model correlation within the OOMI IDE is conducted during the *Register Component Class Representation (CCR)* phase from the Correlator Panel. The Correlator Panel is shown in Figure IV-15 within the dashed rectangular border. The Correlator Panel is implemented as a Java JPanel and is a component of the "Register" tab.

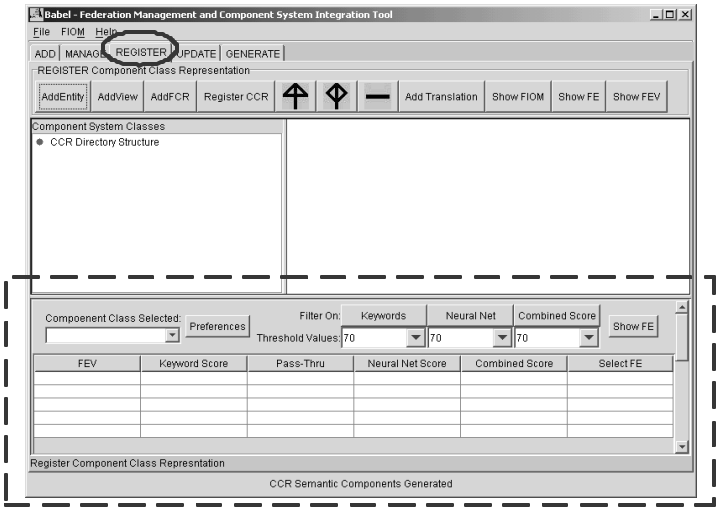


Figure IV-15. The correlator panel of the OOMI IDE.

The Correlator Panel includes a number of features important to the interoperability engineer. The table in the lower portion is used to display the results of the correlation process. The "Preferences" button can be clicked to set various parameters used in the correlation process such as the learning rate, maximum iterations and error tolerance for the neural network training process. Drop down boxes are provided so threshold settings can be set for each phase of the correlation process. The "Keyword" button is used to launch the semantic correlation phase and the "Neural Net" button is used to launch the syntactic correlation phase. The "Combined Score" button filters the correlation results by the combined keyword and neural network score of an FEV.

As stated in Section IV.A.1, the *ComponentModelCorrelator* presented in this thesis implements the *CorrelatorPanelEventListener* interface shown in Figure IV-4. The class *CorrelatorPanelEvent* defines the common events associated with the component model correlator panel within the OOMI IDE. When a feature on the correlator panel is selected it fires a *CorrelatorPanelEvent* instance that is then acted upon by the *ComponentModelCorrelator*. The *CorrelatorPanelEvent* class defines five events:

- PREFERENCES_BUTTON_PRESSED – This event is fired when the "Preferences" button is clicked. The *CorrelatorPanelEventListener* then calls the *panelEventOccurred* method of the *ComponentModelCorrelator*. The

ComponentModelCorrelator handles the event through a switch block and calls the *setPreferences* method.

- KEYWORD_BUTTON_PRESSED – This event is fired when the "Keyword" button is clicked. The *CorrelatorPanelEventListener* then calls the *panelEventOccurred* method of the *ComponentModelCorrelator*. The *ComponentModelCorrelator* handles the event through a switch block and calls the *doSemanticSearch* method. The *doSemanticSearch* method initializes and manages the semantic correlation as described in Section III.E and displays the results in the Correlator Panel results table.

- NEURAL_NET_BUTTON_PRESSED – This event is fired when the "Neural Net" button is clicked. The *CorrelatorPanelEventListener* then calls the *panelEventOccurred* method of the *ComponentModelCorrelator*. The *ComponentModelCorrelator* handles the event through a switch block and calls the *doSyntacticSearch* method. The *doSyntacticSearch* method initializes and manages the semantic correlation as described in Section III.F and displays the results in the Correlator Panel results table alongside the semantic correlation results.

- SHOW_FEV_BUTTON_PRESSED – This event is fired when the "Show FEV" button is clicked. The *CorrelatorPanelEventListener* then calls the *panelEventOccurred* method of the *ComponentModelCorrelator*. The *ComponentModelCorrelator* handles the event through a switch block and calls the *showFEV* method. This method displays a selected results FEV to the display area above the Correlator Panel on the "Register" tab. This allows the interoperability engineer to examine any FEV and make a final determination as to its correspondence to a candidate CCR.

- CCR_SELECTED – This event is fired when the "Keyword" button is clicked. The *CorrelatorPanelEventListener* then calls the *panelEventOccurred* method of the *ComponentModelCorrelator*. The *ComponentModelCorrelator* handles the event through a switch block and calls the *updateCCRList* method. This

method sets the candidate CCR attribute to the CCR selected in the combination box.

2. Semantic Correlator

The *mil.navy.nps.cs.babel.Correlator.semanticSearchEngine* package contains the classes that implement the meat of the semantic correlation process. As shown in Figure IV-16, the package contains only one interface, *SemanticSearchEngine,* which is used by the *ComponentModelCorrelator* to conduct the semantic correlation. The class *SemanticSearchEngineImpl* provides an implementation for the interface.

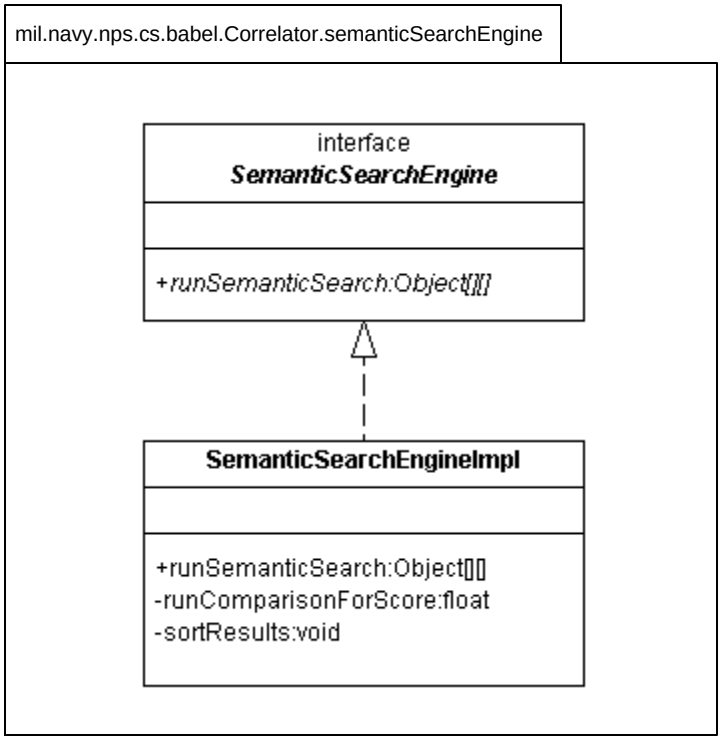


Figure IV-16. UML Class diagram of package *semanticSearchEngine*

The interoperability engineer initiates the semantic correlation process by clicking the “Keywords” button from Correlator Panel as shown in Figure IV-12.

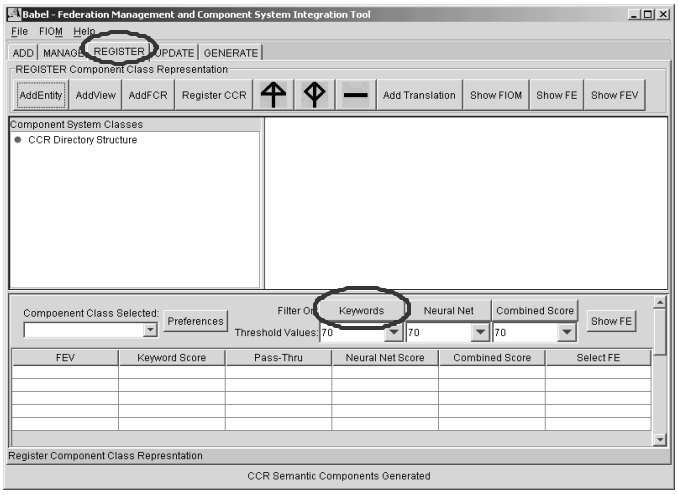


Figure IV-17. Initiating the semantic correlation process.

The *doSemanticCorrelation* method of the *ComponentModelCorrelator* is then called and the semantic correlation process begins. A UML Sequence diagram describing the semantic correlation process is shown in Figure IV-18.

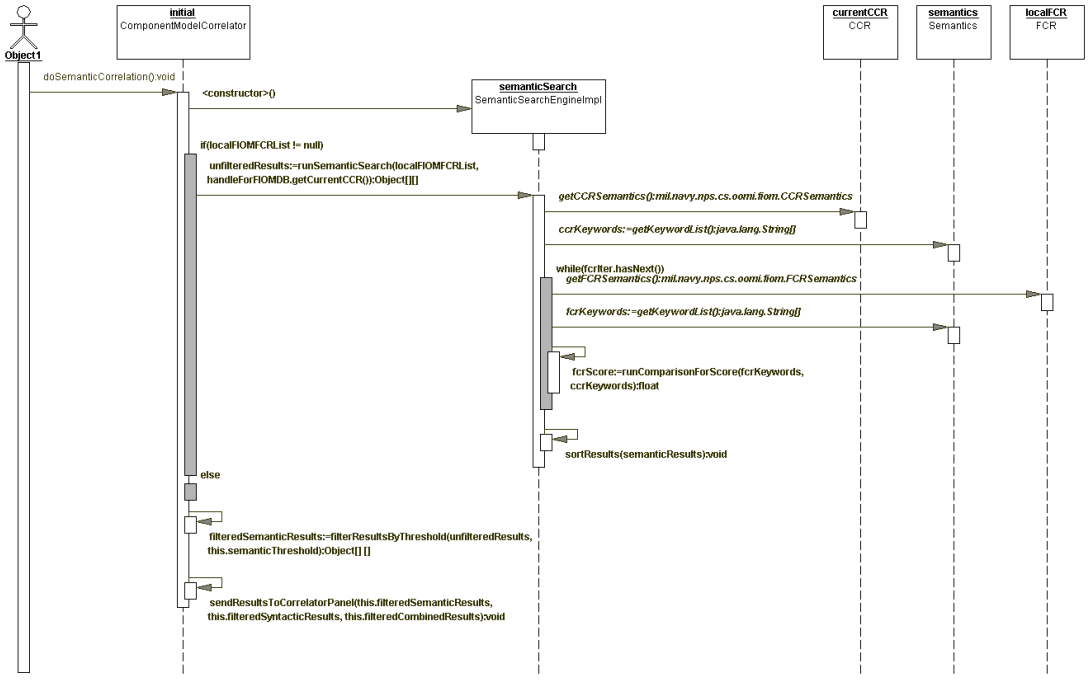


Figure IV-18. UML Sequence diagram showing the semantic correlation

The *doSemanticCorrelation()* method takes control of the semantic correlation process. An instance of *SemanticSearchEngineImpl* is created and the *runSemanticSearch* method is called. The *runSemanticSearch* method takes a list of FCRs as one parameter and the candidate CCR as the second parameter. The *SemanticSearchEngine* then executes the semantic correlation algorithm in accordance with the methodology discussed in Section III.E. Once the semantic correlation is complete, the *SemanticSearchEngineImpl* returns a two dimensional array of results to the *ComponentModelCorrelator*. The two dimensional array contains a mapping for each element consisting of the name of the FCRs evaluated and the associated semantic correlation score for the FCR. These results are then passed to the Correlator Panel and displayed in the results table.

3. Syntactic Correlator

The *mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine* package contains the classes that implement the meat of the syntactic correlation process. As shown in Figure IV-16, the package contains only one interface, *SyntacticSearchEngine,* which is used by the *ComponentModelCorrelator* to conduct the syntactic correlation. The class *SyntacticSearchEngineImpl* provides an implementation for the interface.

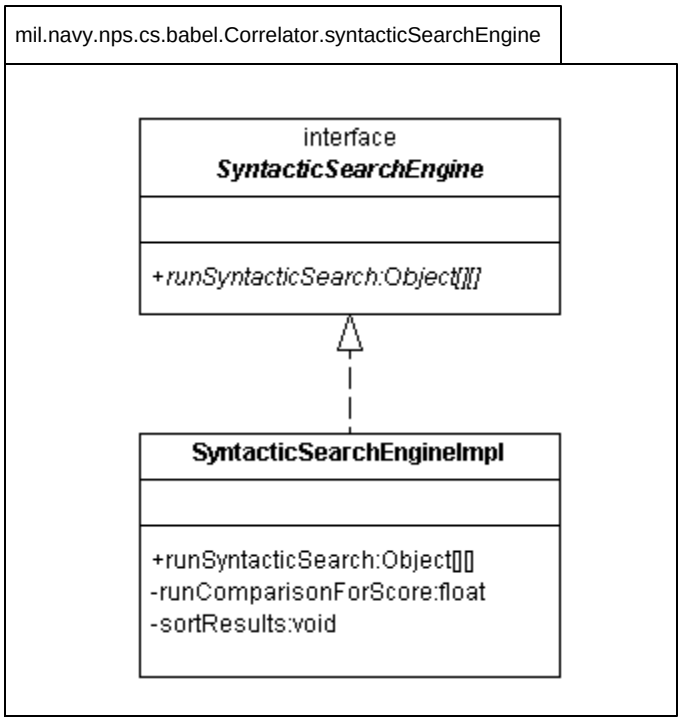


Figure IV-19. UML Class diagram of package *syntacticSearchEngine*

The interoperability engineer initiates the syntactic correlation process by clicking the "Neural Net" button from the "Register" tab in the IDE as shown in Figure IV-12.

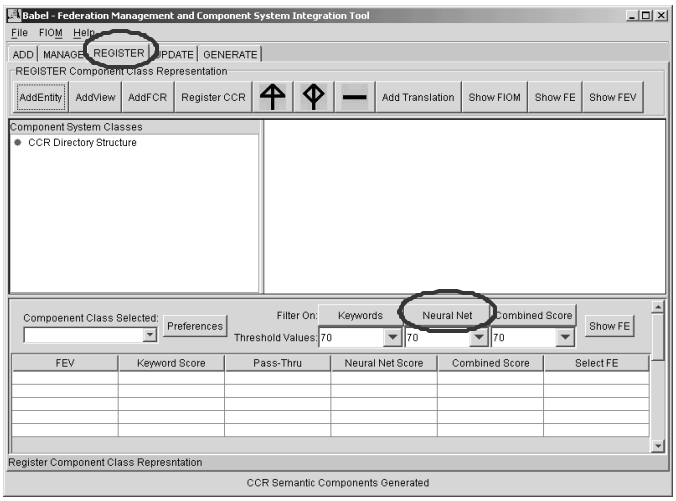


Figure IV-20. Initiating the syntactic correlation process.

The *doSyntacticCorrelation* method of the *ComponentModelCorrelator* is then called and the syntactic correlation process begins. A UML Sequence diagram describing the syntactic correlation process is shown in Figure IV-18.

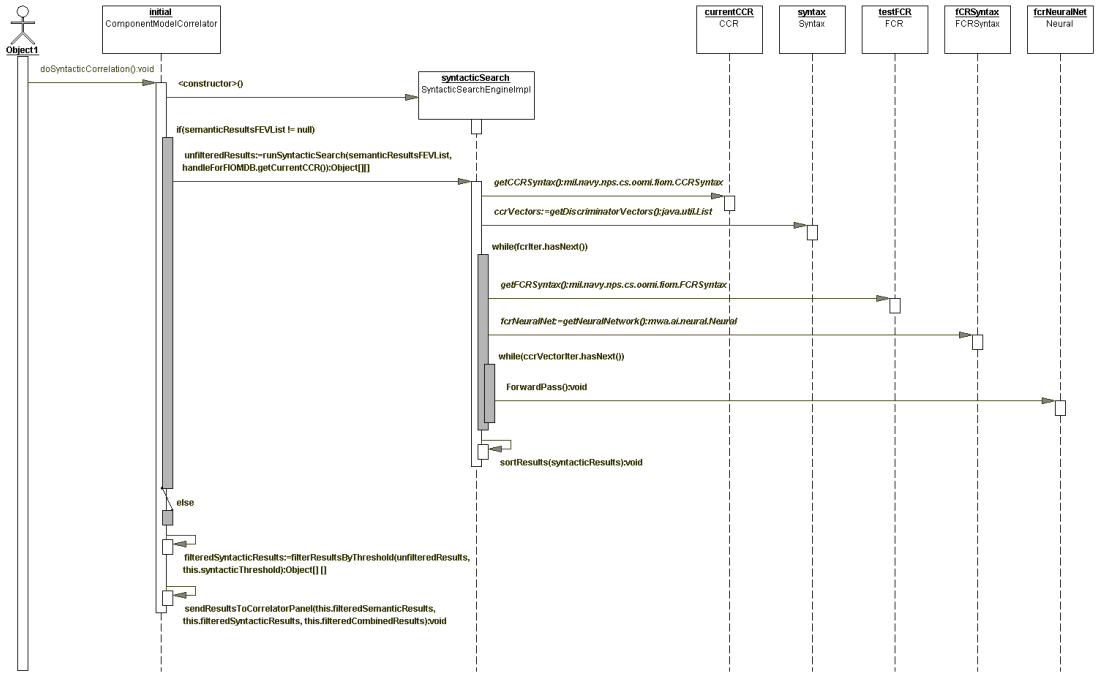


Figure IV-21. UML Sequence diagram showing the syntactic correlation

The *doSyntacticCorrelation* method controls the syntactic correlation process. An instance of the *SyntacticSearchEngine* is created and the *runSyntacticSearch* method is called. The *runSyntacticSearch* method takes a list of FCRs as one parameter and the candidate CCR as the second parameter. The *SyntacticSearchEngine* then executes the syntactic correlation algorithm as discussed in Section III.F. Similar to the semantic search engine, the SyntacticSearchEngineImpl returns a two dimensional array of results to the *ComponentModelCorrelator*. The two dimensional array contains a mapping for each element consisting of the name of the FCRs evaluated and the associated syntactic correlation score for the FCR. These results are then passed to the Correlator Panel and displayed in the results table.

V. CONCLUSIONS

A. PRELIMINARY RESULTS

A small test set of five XML schemas, based on the systems in Figure II-1 of Section II.A.3, was created to test the prototype of the Component Model Correlator. The XML Schemas are listed in Appendix D. Of these five schemas, two define Component Class Representation (CCR) schemas and three define Federation Class Representation (FCR) schemas. The schemas were used by the Component Model Correlator to generate the semantic and syntactic components required for each phase of component model correlation. After the generation of the semantic and syntactic components, each CCR was semantically and syntactically correlated to the set of three FCRs. The results for each phase of the correlation are presented in the following sections. To summarize, the preliminary results are inconclusive and do not provide enough data to evaluate the component model correlation methodology or the Component Model Correlator prototype. The major reason for the inconclusive results is the small set of federation entities created for the tests. In order to better evaluate the Component Model Correlator prototype and the component model correlation methodology, a robust set of federation components is required. The creation of a large test set of federation components is left as a subject for future work.

1. Semantic Correlation Results

The preliminary results of the semantic correlation phase for each CCR are shown in Table V-1. By test design the *armoredFightingVehicle* CCR should correlate with the *groundCombatVehicleView3* FCR. Also, the *mechanizedCombatVehicle* CCR should correlate with the *groundCombatVehicleView1* FCR. As shown, the semantic correlation is successful in correlating the test CCRs to the test design FCRs.

Table V-1. Preliminary semantic correlation results.

CCR	FCR Results
armoredFightingVehicle	groundCombatVehicleView3: 75.4 groundCombatVehicleView2: 67.2 groundCombatVehicleView1: 67.2
mechanizedCombatVehicle	groundCombatVehicleView1: 58.9 groundCombatVehicleView3: 55.2 groundCombatVehicleView2: 55.2

Despite an apparently successful semantic correlation, the percentage results for the *mechanizedCombatVehicle* in Table V-1 are lower than originally anticipated. As a results, an additional test was conducted using a slight modification in the methodology for generating the semantic components discussed in Section III.C. The original methodology uses seven XML Schema fields for keyword generation. These fields and their descriptions are listed in Table III-1. In the second test, five of the seven fields were eliminated from the keyword generator leaving only the *xsd:documentation* and *xsd:enumeration* fields. Table V-2 shows the results of the semantic correlation conducted with the modified semantic components. In this test case, the FCR results for each CCR increase dramatically while still returning the expected design results. Although interesting, the results are inconclusive given the small training set and cannot be used to suggest a change in the methodology of Section III.C. As stated, a larger set of federation components is needed to fully validate the Component Model Correlator.

Table V-2. Semantic correlation results with modified semantic components.

CCR	FCR Results
armoredFightingVehicle	groundCombatVehicleView3: 91.4 groundCombatVehicleView2: 77.1 groundCombatVehicleView1: 77.1
mechanizedCombatVehicle	groundCombatVehicleView1: 97.1 groundCombatVehicleView3: 91.4 groundCombatVehicleView2: 91.4

2. Syntactic Correlation Results

The preliminary results of the semantic correlation phase for each CCR are shown in Table V-3. As with the semantic correlation test, the *armoredFightingVehicle* CCR should correlate with the *groundCombatVehicleView3* FCR and the *mechanizedCombatVehicle* CCR should correlate with the *groundCombatVehicleView1* FCR. As shown, these results were not obtained in the semantic correlation phase. Instead, the *armoredFightingVehicle* syntactically correlated to the *groundCombatVehicleView1* FCR and the *mechanizedCombatVehicle* syntactically correlated to *groundCombatVehicleView3* FCR. The syntactic correlation of Table V-3 did not use the one-to-one correspondence method for reducing false-positives as described in Section III.F. Interestingly, no results are returned for either CCR with the one-to-one correspondence check in place. This means that despite the 95 and 94 percent syntactic correlation scores in Table V-3, the correlation were not one-to-one. Again, this is not the desired result.

Table V-3. Preliminary syntactic correlation results.

CCR	FCR Results
armoredFightingVehicle	groundCombatVehicleView2: 94.2 groundCombatVehicleView1: 93.5 groundCombatVehicleView3: 88.4
mechanizedCombatVehicle	groundCombatVehicleView3: 95.3 groundCombatVehicleView1: 94.1 groundCombatVehicleView2: 82.9

Despite the failure to achieve the expected correlations, the preliminary results for the syntactic correlation are inclusive. It is not clear if the failure is a part of the methodology or a part of the test design. An analysis of the raw discriminator vectors generated by the Syntactic Component Generator suggests that the Syntactic Correlator is performing correctly and that perhaps the test design and test XML schemas are flawed. For example, visual inspection shows that the discriminator vectors for the *mechanizedCombatVehicle* are closer in value to the *groundCombatVehicleView3* FCR than they are to any other FCR. This indicates that the XML Schemas may have been created incorrectly. Although the schemas provide the correct semantic results, they were

not created to give consistent syntactic correlation results. In order to accurately assess the syntactic correlation process and the prototype presented, a robust set of test federation components is required. In addition, syntactic correlation tests must include designed successful correlations and designed failures.

B. NEURAL NETWORK CONSIDERATIONS

As discussed in Section III.D.2, a backpropagation neural network is one of the syntactic components generated for an FCR. During the generation process, the network is trained using the FCR's discriminator vectors and a set of binary output vectors representing the FCR's attributes. The training process for a neural network can be time consuming and resource intensive. Hence, a balance must be achieved between the effort to train a network for correct responses and the effort to train for good responses. Several factors address this balance:

Choice of initial weights – The choice of initial weights will influence whether the net reaches a global minimum of the output error and, if so, how quickly it converges on the desired output. It is important to avoid choices of initial weights that would make it likely that either a nodes activation function or its derivative is zero. Also, the initial values cannot be too low or too high where the values approach the upper and lower bounds of the sigmoid function. With weight values in these regions, training can be very slow. [Fau94]

Learning rate – The learning rate is a parameter that controls the amount by which weights are changed during training. In a backpropagation neural network, the learning rate remains constant throughout the training process. A low learning rate increases the time required to train the network since the adjustments to weights occur in small increments. However, the lower learning rate usually results in a lower output error and better-trained network. Conversely, a high learning rate will decrease training time but result in a higher output error. [Fau94]

Maximum Error Tolerance – The maximum error tolerance is the maximum valued for the mean squared error of the neural network output during training. The mean squared error is defines as the sum of the squared node output errors divided by the

number of input training patterns. If the maximum error tolerance is set too low, the training cycle may take a long time to train, although the trained network will produce better results. Conversely, if the error tolerance is set too high, the network will train faster, but with less accurate matching capability. [Fau94] The maximum error tolerance and maximum epochs factor work in conjunction with each other to limit the training time for a neural network.

Node input bias – A node input bias can assist the network in training when many of the input training values are at the bound of the sigmoid function, which for an FCR is either 0.0 or 1.0. The node input bias is a term which is included in the net input for node j along with the sum of all weighted inputs from all nodes connected to node j. This provides an input to node j that can be propagated with the nodes activation function. [Fau94]

Maximum epochs – An epoch is one presentation of the set of input training vectors to the neural network. Since the network must produce the set of desired outputs for the set of training input, cycles of training are characterized as epochs. The maximum number of epochs setting limits the training epochs for a neural network should the maximum error tolerance not be achieved.

The implementation of the syntactic component generator presented in Section IV.B.3, addresses neural network training factors in a cursory manner in order to prove the concept of syntactic correlation. The value or method used for each factor is shown in Table V-4. In future work, the Component Model Correlator should be revised to optimize these neural network training factors.

Table V-4. Neural network training factors used in the Component Model Correlator implementation.

Factor	Value/Method
Initial Weight Values	Random numbers in range [-0.5, 0.5]
Learning rate	0.25
Max Error tolerance	0.01
Bias	Not used
Max Epochs	100,000

C. THE ROLE OF COMPONENT MODEL CORRELATION

The component model correlation methodology and component model correlator presented in this thesis provide a viable solution to model correspondence within the OOMI IDE. The introduction of automated component model correlation is a significant benefit of the OOMI IDE over other methods of interoperability such as CORBA or Java Enterprise. [You02] Correlation automation saves interoperability engineers countless man-hours during the process of integrating systems and enables new components to be added to a federation in a relatively short amount of time.

The two-phase approach to correlation provides mechanisms to increase the recall and precision of the search space. The semantic search process capitalizes from a straightforward classical keyword matching approach that allows for flexibility and adaptation. Improvements or changes to the keyword search algorithm can be incorporated into the component model correlator with ease. The syntactic search process utilizes advanced artificial intelligence methods to compare the structure of component and federation data models to locate an entity correspondence. In varying implementations, these two processes can be adapted for multi-threading or can be placed in a pipeline architecture with little change to the underlying code.

With the component model correlator module, the Object-Oriented Method for Interoperability is a step in the right direction in providing a concrete methodology and

framework to make interoperability a reality, not just a concept in a white paper. This moves the Department of Defense closer to the Quadrennial Defense Review goal of having systems that can immediately "plug" into other joint battlefield operating systems.

D. RECOMMENDATIONS FOR FUTURE RESEARCH

At the time of this writing, the OOMI IDE toolset is far from completion. The Interoperability Working Group at the Naval Postgraduate School continues to develop the features of the IDE and research future directions. One item of discussion is the possibility of using a native XML database to store FIOM objects. Another topic is the use of XML data binding and Java reflection to build the semantic and syntactic components rather than parsing through an XML Schema. An exciting future area of research is the idea of a distributed OOMI IDE, which would distribute the components of the FIOM to clusters of users.

1. Test of Component Model Correlator with a large set of federation components.

As discussed in Section V.A, the preliminary test results for the semantic and syntactic correlation implemented in the Component Model Correlator prototype are inconclusive and do not provide enough data to evaluate the methodology or the prototype. Two possible reasons for the inconclusive result are the small number of federation test components and the possible flawed construction of the XML test Schemas. In order to accurately assess the Component Model Correlator methodology, a large set of federation components is required. The testing of the correlator methodology and prototype would be extremely valuable in accurately assessing the benefits of the Component Model Correlator.

2. XML Documents for Semantic and Syntactic Components.

The use of XML documents to provide persistent storage for semantic and syntactic components used for the component model correlator is a natural improvement over the implementation described in this thesis. For example, the keyword lists and discriminator vectors for a CCR and FCR can easily be stored in XML documents. Similarly, the neural network file can be changed from the format described in Section IV.F to a more streamlined XML format. Future work could address the use of XML

documents and a native XML database to store the objects of an FIOM. An example of a native XML database is the Apache Xindice project. [XIN02] A benefit of using a native XML database is the reduction in the amount of work required to map XML to some other data structure. Instead, data is inserted into the database as XML and retrieved as XML. There is also a lot of flexibility through the semi-structured nature of XML and the schema independent model used by Xindice. This is especially valuable with very complex XML structures that would be difficult or impossible to map to a more structured database.

3. Semantic and Syntactic Component Generation Using Java Reflection

In the original component model correlation methodology described by Young, semantic and syntactic components were generated from FCRSchema and CCRSchema classes defined within the FIOM Framework. [You02] These Java objects are created using XML data binding. XML data binding is the process of mapping an XML document to its in-memory object representation. The conversion of an XML document into Java objects is called unmarshalling. In theory, once a Java object(s) exist for an XML document, Java reflection could be used to capture the semantic and syntactic information required by the component model correlator. Current methods for conducting XML data binding do not capture the full spectrum of data contained in an XML Schema. Future improvements to XML data binding technology might enable the use of Java reflection. Future work could refine the use of Java reflection in order to create the semantic and syntactic components directly from memory rather than from XML Schemas.

4. Object Correlation in a Distributed OOMI IDE Architecture

A distributed OOMI IDE introduces many complex issues into an already complex tool set. Component model correlation in a distributed environment may or may not be a problem. In one view, the component model correlator module could reside on each system within the distributed environment. When correlation is performed, the requesting system could send a request to the distributed correlator modules to find FCR matches. In this type of distributed environment, the features of the component model correlator would need only minor improvements given their low coupling to the OOMI IDE. In another

view, the distributed OOMI IDE would still require an executive that would oversee all correlation between the various systems. In this case, the correlator module may need more rework. Future work in this arena could include the design and implementation of a distributed Component Model Correlator.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[Ant94] Anton, H., *Elementary Linear Algebra*, John Wiley & Sons, Inc., NewYork, NY, 1994.

[BM01] Biron, P., Malhotra, A. (editors), "XML Schema Part 2: Datatypes", [http://www.w3.org/TR/xmlschema-2/], 2 May 2001.

[CJCS01] Chairman of the Joint Chiefs of Staff, "Charter of the Joint Requirements Oversight Council," CJCSI 5123.01A, Washington, D.C., March 2001.

[Fau94] Fausett, L., *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice Hall, 1994.

[HCD+01] Hunter, D., Cagle, K., Dix, C., Kovack, R., Pinnock, J., Rafter, J., *Beginning XML*, 2ed, Wrox Press, Birmingham, UK, 2001.

[Hay94] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing, New York, 1994.

[HH02] Le Hors, A., Le Hégaret, P. (editors), "Document Object Model (DOM) Level 3 Core Specification", [http://www.w3c.org/TR /2002/WD-DOM-Level-3-Core-20020409/], 4 April 2002.

[HL96] Holowczak, R., Li, W., "A Survey on Attribute Correspondence and Heterogeneity Metadata Representation", [http://www.computer.org/conferences/meta96/li/paper.html], 1996.

[HM02] Hunter, J., McLaughlin, B., "JDOM Frequently Asked Questions", [http://www.jdom.org/docs/faq.html], 2002.

[LC00] Li, W., Clifton, C, "SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks.," Data and Knowledge Engineering, Vol. 33 (2000), pp.49-84.

[Lee02] Lee, S., *Class Translator for the Federation Interoperability Object Model (FIOM)*, Master's thesis, Naval Postgraduate School, Monterey, California, March 2002.

[Pug01] Pugh, R.G., *Methods For Determining Object Correspondence During System Integration*, Master's Thesis, Naval Postgraduate School, 2001.

[QDR01] *Quadrennial Defense Review*, Department of Defense, September 30, 2001.

[SL98] Stubblefield, W. and Luger, G., *Artificial Intelligence*, Addison-Wesley, 1998.

[SM83] Salton, G., McGill, M., *Introduction to Modern Information Retrieval*. McGraw Hill, New York, 1983.

[TBM+01] Thompson, H., Beach, D., Maloney, M., Mendelsohn, N. (editors), "XML Schema Part 1: Structures", [http://www.w3.org/TR/xmlschema-1/], 2 May 2001.

[Wat97] Watson, M., *Intelligent Java Applications for the Internet and Intranets*, Morgan Kaufmann, 1997.

[Wie93] Wiederhold, G., "Intelligent Integration of Information", *ACM-SIGMOD93*, Washington, DC, May 1993, pp. 434-437.

[XIN02] "About Apache Xindice." [http://xml.apache.org/xindice/index.html]

[You02] Young, P.E., *Heterogeneous Software System Interoperability Through Computer-Aided Resolution of Modeling Differences*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, 2002.

APPENDIX A. COMPONENT MODEL CORRELATOR SOURCE

A. PACKAGE: mil.navy.nps.cs.babel.correlator.semanticComponentGenerator

1. Interface KeywordGenerator

```
//*****************************************************************************
// Filename:.............KeywordGenerator.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.semanticComponentGenerator;

import org.jdom.JDOMException;
import org.jdom.Document;


/** ************************************************************************
 * <br>
 * The <i>KeywordGenerator</i> interface defines the services that a keyword
 * generator must provide to the component model correlator.  The class
 * <i>ComponentModelCorrelator</i> uses this interface to instantiate the keyword
 * generator.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 ************************************************************************** */
public interface KeywordGenerator
{

   /** ************************************************************************
    * <br>
    * Returns an array of keywords extracted from the JDOM document
    * passed in as a parameter.
    *
    * @param jdom - A JDOM Document created from a well-formed XML Schema
    * @return A String array containing the keywords from the JDOM Document.
    * @throws JDOMException - if the jdom parameter is null or if there are any
    * other JDOM problems.
    *
    * @author LT Steve Shedd
    *
    ************************************************************************** */
   public String[] generateKeywords( Document jdomDoc ) throws JDOMException;


}      // End KeywordGenerator
```

2. Class KeywordGeneratorImpl

```
//*****************************************************************************
// Filename:.............KeywordGeneratorImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
```

```
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.semanticComponentGenerator;

import java.util.List;
import java.util.Iterator;
import java.util.Vector;
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.Arrays;
import java.util.TreeSet;
import java.util.Comparator;

import org.jdom.JDOMException;
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.transform.JDOMSource;
import org.jdom.input.SAXBuilder;
import org.jdom.Element;
import org.jdom.Attribute;
import org.jdom.Namespace;


/** ***************************************************************************
 * <br>
 * The <i>KeywordGeneratorImpl</i> class extracts keyword information from an XML
 * document passed into the <i>generateKeywords</i> method.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 **************************************************************************** */
public class KeywordGeneratorImpl implements KeywordGenerator
{

   /** Member variable to hold the JDOM Document used in the parsing. */
   private Document jdomDoc = null;


   /** Data structure to build a raw keyword list */
   private StringBuffer wordList = new StringBuffer();


   /** Final array of keywords generated from the JDOM Document */
   private String[] keywordList = null;


   /** Root Element of the JDOM Document */
   protected Element docRoot = null;


   /** Namespace used within the JDOM Document */
   Namespace docNamespace = null;


   /** Default constructor */
   public KeywordGeneratorImpl()
   {

   }


   /** ************************************************************************
    * <br>
    * Returns an array of keywords extracted from the JDOM document passed in as
    * a parameter.  The method is a requirement from the interface
    * <i>KeywordGenerator</i>.
    *
    * @param jdom - A JDOM Document created from a well-formed XML Schema
```

```
 * @return A String array containing the keywords from the JDOM Document.
 * @throws JDOMException - if the jdom parameter is null or if there are any
 * other JDOM problems.
 *
 * @author LT Steve Shedd
 * @version 1.0
 *
 ************************************************************************** */
public String[] generateKeywords( Document jdom ) throws JDOMException
{

   // reset the results variables
   wordList = new StringBuffer();
   keywordList = null;

   // Check to see if a null Document has been passed into the method
   if ( jdom == null )
   {

      throw new JDOMException( "Null pointer to JDOM Document",
      new NullPointerException() );

   }

   // Create a local copy of the JDOM Document
   this.jdomDoc = new Document( jdom.getContent() );

   // Identify root element of JDOM document and get the schema namespace
   this.docRoot = jdom.getRootElement();
   this.docNamespace = this.docRoot.getNamespace();

   // Call the recursive function parseNode with JDOM root element.
   parseNode( this.docRoot );

   // Filter the raw keyword results to get rid of clutter words and other
   // bogus characters.
   String[] results = filterRawKeywords( this.wordList );

   // Return the filtered list of keywords
   return results;


}   // End generateKeywords


/** ************************************************************************
 * <br>
 * A recursive function that parses through the elements of the
 * JDOM Document. The
 * initial element parameter should be the document root of the
 * JDOM document. The
 * method will then recursively walk through the children of each
 * element and call
 * the <i>evaluateNode</i> method.
 *
 * @param root - a root element of the JDOM document.
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0
 *
 ************************************************************************** */
private void parseNode( Element root )
{

   // Get a list of children under the root element parameter
   List kids = root.getChildren();

   // Get an iterator for the kids list
```

```
   Iterator kidsIter = kids.iterator();

   while ( kidsIter.hasNext() )
   {
      // for each child, extract the JDOM Element from the list that
      // represents the child.
      Element childNode = ( Element )kidsIter.next();

      // Evaluate the element for keywords
      evaluateNode( childNode );

      // then call the recursive method to look for grandchildren
      parseNode( childNode );

   }

}   // End parseNode


/** ****************************************************************************
 * <br>
 * Evaluates a node of the JDOM document and determines whether or not
 * to add keywords to the components keyword list.  This method implements
 * the logic of the keyword generator as defined in the semantic component
 * generation methodology.
 *
 * @param node - A JDOM element from which keywords may be extracted.
 * @return None
 *
 * @author LT Steve Shedd
 * @version 1.0
 *
 **************************************************************************** */
private void evaluateNode( Element node )
{

   String localName = node.getName();

   // The if blocks below check the type of JDOM element being evaluated.
   // For each type, specific attribute keywords are extracted.  Or, in the
   // case of the documentation element, the contents of the element are
   // extracted.  A space is added to the end of each append call in order
   // to facilitate filtering in the filterRawKeywords method.

   if ( localName.toLowerCase().equals( "element" ) )
   {
      this.wordList.append( node.getAttributeValue( "name" ) + " " );
      this.wordList.append( node.getAttributeValue( "type" ) + " " );
      this.wordList.append( node.getAttributeValue( "ref" ) + " " );
   }

   if ( localName.toLowerCase().equals( "documentation" ) )
   {
      this.wordList.append( node.getTextNormalize() + " " );
   }

   if ( localName.toLowerCase().equals( "enumeration" ) )
   {
      this.wordList.append( node.getAttributeValue( "value" ) + " " );
   }

   if ( localName.toLowerCase().equals( "attribute" ) )
   {
      this.wordList.append( node.getAttributeValue( "name" ) + " " );
   }

   if ( localName.toLowerCase().equals( "simpleType" ) )
   {
      this.wordList.append( node.getAttributeValue( "name" ) + " " );
```

```
   }


}   // end of evaluateNode


/** ****************************************************************************
 * <br>
 * Returns a filtered list of keywords extracted from the
 * XMLSchema document. This
 * method is intended to be used to filter out clutter words and
 * characters from the
 * raw string buffer. This version performs simple filtering for simple
 * characters and a
 * small set of noise words.  This method should be developed to
 * include more common
 * english words that could be considered clutter.    *
 *
 * @param words - a StringBuffer containg all the keywords extracted
 * from an XML Schema.
 * @return An array of individual keywords.
 *
 * @author LT Steve Shedd
 * @version 1.0
 *
 ***************************************************************************** */
private String[] filterRawKeywords( StringBuffer rawWords )
{

   // Need a temporary data structure that can grow dynamically.  The
   // TreeSet class gives the added benifit of eliminating duplicate
   // values and conducting an insertion sort on new elements.
   TreeSet tempResults = new TreeSet();

   // The final array that will be passed back to the caller
   String[] results = null;

   // Convert the input string buffer to one large string
   // in order to make use of the String.replace method
   String words = rawWords.toString();
   words = words.replace( '_', ' ' );   // remove underbars
   words = words.replace( ':', ' ' );   // remove colons
   words = words.replace( ';', ' ' );   // remove semi-colons
   words = words.replace( '"', ' ' );   // remove all double quotation marks
   words = words.replace( '.', ' ' );   // remove all periods
   words = words.replace( ',', ' ' );   // remove all commas
   words = words.replace( '(', ' ' );   // remove all left parenthesis
   words = words.replace( ')', ' ' );   // remove all right parenthesis
   words = words.trim();                // remove spaces at beginning and end

   // Create a string tokenizer so we can extract individual
   // keywords from the words string.
   StringTokenizer st = new StringTokenizer( words );

   // Iterate through the String and extract individual keywords
   while ( st.hasMoreTokens() )
   {
      // Add the word to our list
      tempResults.add( st.nextToken() );
   }


   // The next section of code checks the words in the temp results to
   // see if we should exclude any from the final results.

   Iterator tempResultsIter = tempResults.iterator();
   int counter = tempResults.size();

   while ( tempResultsIter.hasNext() )
```

```
   {

      String word = ( String )tempResultsIter.next();

      if ( inExclusionList( word ) )
      {
         tempResults.remove( word );

         // Since there has been a change to the list, we
         //need to get a new iterator
         tempResultsIter = tempResults.iterator();
      }

   }

   // Size the final results String array
   results = new String[ tempResults.size() ];

   // Convert the temporary list of keywords to an array of strings
   results = ( String[] ) tempResults.toArray( results );

   // return the sorted string array of keywords
   return results;

}   // End filterRawKeywords


/** ****************************************************************************
 * <br>
 * Returns true if the word to check is in the exclusion list,
 * false otherwise. The
 * exclusion in this version is only an example.  A deplyable version
 * would most likely
 * open an exculsion list file that would contain thousands of words
 * that would clutter
 * a list of keywords.
 *
 * @param wordToCheck - the string to search for in the exclusion list.
 * @return True if the word is in the exclusion list, false otherwise.
 *
 * @author LT Steve Shedd
 * @version 1.0
 *
 ******************************************************************************* */
private boolean inExclusionList( String wordToCheck )
{

   boolean result = false;

   // Very small example set of words to exclude from the keyword list.
   // If the list is larger, it would be better to use a binary search
   // tree as the data structure.
   String[] exclusionList = { "and", "but", "or", "nor", "in", "on", "while",
                              "how", "when", "where", "why", "i", "were", "was",
                              "is", "has", "had", "of", "a", "the", "to", "null",
                              "for", "an", "this", "that", "which", "what", "as"};

   for ( int i = 0; i < exclusionList.length; i++ )
   {
      if ( ( ( String )exclusionList[i] ).equalsIgnoreCase( wordToCheck ) )
      {
         result = true;
         break;
      }
   }

   return result;

}   // End inExclusionList
```

```
}       // End KeywordGeneratorImpl
```

B. PACKAGE: mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator

1. Interface DiscriminatorGenerator

```
//*****************************************************************************
// Filename:.............DiscriminatorGenerator.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator;

import java.util.List;

import org.jdom.JDOMException;
import org.jdom.Document;


/** ************************************************************************
 * <br>
 * The <i>DiscriminatorGenerator</i> interface defines the services
 * that a discriminator
 * vector generator must provide to the component model correlator.  The class
 * <i>ComponentModelCorrelator</i> uses this interface to build
 * discriminator vectors
 * for use in the syntactic correlation process
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 ************************************************************************** */
public interface DiscriminatorGenerator
{


   /** ************************************************************************
    * <br>
    * Returns a list of discriminator vectors constructed from the JDOM document
    * passed in as a parameter.  Each element of the list is a 28
    * element float array
    * containing the discriminant values defined in the syntactic
    * component generation
    * methodology.
    *
    * @param jdom - A JDOM Document created from a well-formed XML Schema
    * @return A List of discriminator vectors for the component.
    * @throws JDOMException - if the jdom parameter is null or if there are any
    * other JDOM problems.
    *
    * @author LT Steve Shedd
    *
    ************************************************************************** */
   public List generateDiscriminatorVectors( Document jdomDoc ) throws JDOMException;


}       // End DiscriminatorGenerator
```

2. Class DiscriminatorGeneratorImpl

```
//*********************************************************************************
// Filename:.............DiscriminatorGeneratorImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*********************************************************************************


package mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator;

import java.util.List;
import java.util.Iterator;
import java.util.Vector;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;

import java.io.File;
import java.io.IOException;

import org.jdom.JDOMException;
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.transform.JDOMSource;
import org.jdom.input.SAXBuilder;
import org.jdom.Element;
import org.jdom.Attribute;
import org.jdom.Namespace;


/** ********************************************************************************
 * <br>
 * The <i>DiscriminatorGeneratorImpl</i> generates a set of discriminator
 * vectors from a JDOM Document representation of a well formed XML
 * Schema.  The discriminator vectors are then used in the syntactic
 * correlation process.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *********************************************************************************
@testcase
test.mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator.TestDiscriminatorGenera
torImpl
 *
 ********************************************************************************* */
public class DiscriminatorGeneratorImpl implements DiscriminatorGenerator
{

   //Generator Control Variables


   /** ********************************************************************************
    *  A copy of the JDOM document created by babel to process CCR
    *  and FCR XML Schemas.
    ********************************************************************************* */
   private Document jdomDoc = null;


   /** ********************************************************************************
    *  An array list type that contains the elements of the XML schema
    *  in a type heirarchy.  An array list was chosen over a Tree because
    *  of the ability to store lists within lists and benefit from
    *  recursion for some utitily methods.  The ArrayList data type can
    *  also contain null values.  Many other data structures
```

```
 *  could be used for this purpose, however, I decided to go with an
 *  array list.
 *
 *  The adopted convention for using the ArrayList in
 *  the DiscriminatorGeneratorImpl class is as follows:
 *
 *  List = { Element Name, Element Discriminator Vector [, children ] }
 *
 *  or in terms of types:
 *
 *  List = { String, DiscriminatorVector [, ArrayLists ] }
 *
 *  The ordering of these elements is critical since the get method of
 *  ArrayList is used extensively.  The get method takes an integer index
 *  as its parameter and returns the object at that index.  Therefore,
 *  the node type name will always be the first index, the associated
 *  discriminator vector at the second index.  Corresponding children
 *  start in the third index and continue as needed.
 *
 ********************************************************************************* */
private ArrayList schemaTypeHierarchy = null;


/** *********************************************************************************
 *  This hash map is legend between simple data types defined in the
 *  JDOM document and their constructed discriminator vector.  The key
 *  field of the table is name of the simple data type as specified in
 *  the JDOM document. The value field is the actual discriminator vector
 *  object constructed for the simple type.
 ********************************************************************************* */
private HashMap simpleTypes = null;


/** *********************************************************************************
 *  This hash map stores the qualified names of the complex types in
 *  XML Schema and the associated element.
 ********************************************************************************* */
private HashMap complexTypes = null;


/** Root Element of the JDOM Document and XML Schema */
private Element docRoot = null;


/** Namespace of the JDOM Document and XML Schema */
private Namespace docNamespace = null;


/** Target namespace defined in the root element */
private Namespace targetNamespace = null;


/** Default constructor */
public DiscriminatorGeneratorImpl()
{
   this.init();
}

/**
 * Initialization function to reset all class variables.  Since class
 * scope variables are used for the discriminator generator, we need to
 * reset the variables each time the constructor or
 * generateDisciminatorVectors is called.  In the future, class variables
 * should be eliminated.
 *
 * @author LT Steve Shedd
 *
 */
private void init()
```

```
{
   this.jdomDoc = null;
   this.schemaTypeHierarchy = new ArrayList();
   this.simpleTypes = new HashMap();
   this.complexTypes = new HashMap();
   this.docRoot = null;
   this.docNamespace = null;
   this.targetNamespace = null;
}


/** ****************************************************************************
 * <br>
 * Returns a list of discriminator vectors constructed from the JDOM document
 * passed in as a parameter.  Each element of the list is a 28
 * element float array
 * containing the discriminant values defined in the syntactic
 * component generation
 * methodology. This method is the only visible work horse of the
 * DiscriminatorGeneratorImpl Class.
 *
 * @param jdom - A JDOM Document created from a well-formed XML Schema
 * @return A List of discriminator vectors for the component.
 * @throws JDOMException - if the jdom parameter is null or if there are any
 * other JDOM problems.
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public List generateDiscriminatorVectors( Document jdom ) throws JDOMException
{

   if ( jdom == null )
   {
      throw new JDOMException( "Null pointer to JDOM Document",
      new NullPointerException() );
   }

   // reinitialize all class variables
   this.init();

   // Create a local copy of the JDOM Document
   this.jdomDoc = new Document( jdom.getContent() );

   // Identify root element of JDOM document and get the schema namespace
   this.docRoot = jdom.getRootElement();
   this.docNamespace = this.docRoot.getNamespace();

   // Get a list of additional namespaces so that we can identify a target
   // namespaces if declared.
   List otherNamespaces = this.docRoot.getAdditionalNamespaces();

   Iterator it = otherNamespaces.iterator();
   int count = otherNamespaces.size();

   if ( count > 2 )
   {
      throw new JDOMException( "Schema contains more than one target " +
      "namespace and cannot be processed by the " +
      " Discriminator Generator." );
   }

   while ( it.hasNext() )
   {

      Namespace otherNS = ( Namespace )it.next();

      if ( !otherNS.getPrefix().equals( this.docNamespace.getPrefix() ) )
      {
```

```
            targetNamespace = otherNS;
         }

      }

      // Get the first element under the Schema root.  This is the element that
      // will define the top level sequence of data types for the entire Schema.
      Element schemaDef = this.docRoot.getChild( "element", this.docNamespace );

      // Get the fully qualified name of the schema type
      String schemaDefName = this.makeQualifiedElementName( schemaDef.getAttributeValue(
"name" ),
      this.targetNamespace.getPrefix() );


      // Extract the simple types and associated discriminator Vectors
      this.extractSimpleTypes();

      // Extract the complex types and elements
      this.extractComplexTypes();


      this.schemaTypeHierarchy.add( 0, schemaDefName );
      this.schemaTypeHierarchy.add( 1, new DiscriminatorVectorImpl() );

      // Recursive function to parse through top level elements and
      // build a list hierarchy of data types.
      this.parseElement( this.schemaTypeHierarchy, schemaDefName );

      //Calculate the top level discriminator for the entire class if necessary
      //DiscriminatorVector topDV = (DiscriminatorVector)schemaTypeHierarchy.get(1);

      try
      {
         // Recursively merge the vectors from the simpletype hierarchy
         this.mergeVectors( this.schemaTypeHierarchy );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }

      //Create a list of float arrays for top level discriminator vectors
      List results = new ArrayList();

      for ( int i = 2; i < this.schemaTypeHierarchy.size(); i++ )
      {

         ArrayList temp = ( ArrayList )this.schemaTypeHierarchy.get( i );

         DiscriminatorVector result =
         new DiscriminatorVectorImpl( ( DiscriminatorVector )temp.get( 1 ) );

         float[] local = result.ValuesAsFloatArray();

        /* System.out.print( "\nRaw" + ( i - 1 ) + ": " );

         for ( int j = 0; j < local.length ; j++ )
         {
            System.out.print( local[j] + " " );
         }

         System.out.print( "\n" );*/

         try
         {
            result.normalizeValues();
         }
         catch ( Exception e )
```

```
      {
         System.out.println( e.getMessage() );
         e.printStackTrace();
      }

      // Convert the singleton discriminator vector into an array of floats
      float[] values = result.ValuesAsFloatArray();

      // Add the float array to the final results list
      results.add( values );

   }

   return results;

}   // End generateDiscriminatorVectors


/** ***************************************************************************
 * <br>
 * Recursive function that traverese the JDOM Document and analyzes each element
 * for discriminator values.
 *
 * @param parentList - a layer of the hierarchical list containing all
 * the complex types.
 * @param elementName - the name of the element to be analzed and parsed.
 * @return None
 *
 * @author LT Steve Shedd
 *
 **************************************************************************** */
private void parseElement ( ArrayList parentList, String elementName )
{

   Element startNode = null;
   Element complexNode = null;

   //Get the element node with the name equal to the name in
   //the list of Complex Types
   startNode = ( Element )this.complexTypes.get( elementName );

   if ( startNode == null )
   {
      String msg = "Unable to locate element " + elementName;
      msg = msg + " in complexTypes HashMap.";

      throw new NullPointerException( msg );
   }


   if ( startNode.getName().equals( "element" ) )
   {
      complexNode = startNode.getChild( "complexType", this.docNamespace );
   }
   if ( startNode.getName().equals( "complexType" ) )
   {
      complexNode = startNode;
   }

   // Get the sequence node under the complex node
   Element sequenceNode = complexNode.getChild( "sequence", this.docNamespace );

   // Get the children elements of the sequence
   List sequenceElements = sequenceNode.getChildren( "element", this.docNamespace );

   // Build an iterator for the sequenceElement List
   Iterator sequenceIterator = sequenceElements.iterator();

   // Step through the sequence defined in the Schema
```

```
while ( sequenceIterator.hasNext() )
{

   // Get the next element node defined in the sequence
   Element temp = ( Element )sequenceIterator.next();

   // Create a default DiscriminatorVector that can be used for each child
   DiscriminatorVector dv = null;
   ArrayList newList = new ArrayList();

   String typeName = null;

   if ( temp.getAttribute( "type" ) != null )
   {
      typeName = temp.getAttributeValue( "type" );
   }
   else if ( temp.getAttribute( "ref" ) != null )
   {
      typeName = temp.getAttributeValue( "ref" );
   }


   newList.add( 0, typeName );


   // Set the DiscriminatorVector for this element
   if ( this.simpleTypes.containsKey( typeName ) )
   {
      // get discriminator vector from simpleTypes hashmap
      dv = ( DiscriminatorVectorImpl )this.simpleTypes.get( typeName );
   }
   else
   {
      // Create a new DiscriminatorVector for a complexType
      dv = new DiscriminatorVectorImpl();
   }

   // Check for minOccurs value
   if ( temp.getAttribute( "minOccurs" ) != null )
   {
      double val = Double.parseDouble( temp.getAttributeValue( "minOccurs" ) );
      dv.setDVMinOccurs( val );
   }

   // Check for maxOccurs value
   if ( temp.getAttribute( "maxOccurs" ) != null )
   {
      // If the maxOccurs is specified, set the corresponding
      // property in the discriminator vector.
      double val = Double.parseDouble( temp.getAttributeValue( "maxOccurs" ) );
      dv.setDVMaxOccurs( val );
   }
   else
   {
      // If the maxOccurs is not specified, set the property
      // to a raw value of 1.  This value will be normalized.
      dv.setDVMaxOccurs( 1 );
   }

   // Add the discriminator vector to the newList created for the type
   newList.add( 1, dv );

   // Set the DiscriminatorVector for this element
   if ( !this.simpleTypes.containsKey( typeName ) )
   {
      // If this is a complexType, make a recursive call to parseElement()
      parseElement( newList, typeName );
   }
```

```
      // Add the child types list ("newList") to the end of the parent list.
      // This effectively creates a parent/child relationship that is
      // stored in the parent list.
      parentList.add( newList );

   }

}   // End parseElement



/** ****************************************************************************
 * <br>
 * Utility method that extracts all the complex types in the JDOM
 * Document and stores
 * the names and objects in a hashtable.
 *
 * @param None
 * @return None
 *
 * @author  LT Steve Shedd
 *
 ***************************************************************************** */
protected void extractComplexTypes()
{

   // Build a list of complex elements from element nodes or complex type nodes.
   // This list is simple a look-up reference to help the parseElement method.
   List complexOne = this.docRoot.getChildren( "element", this.docNamespace );
   List complexTwo = this.docRoot.getChildren( "complexType", this.docNamespace );

   Iterator complexOneIterator = complexOne.iterator();

   while ( complexOneIterator.hasNext() )
   {
      Element hashKey = ( Element )complexOneIterator.next();

      // Get the fully qualified name of the type defined in the element
      String hashValue = this.makeQualifiedElementName(
      hashKey.getAttributeValue( "name" ),
      this.targetNamespace.getPrefix() );

      complexTypes.put( hashValue, hashKey );
   }

   Iterator complexTwoIterator = complexTwo.iterator();

   while ( complexTwoIterator.hasNext() )
   {
      Element hashKey = ( Element )complexTwoIterator.next();

      // Get the fully qualified name of the type defined in the element
      String hashValue = this.makeQualifiedElementName(
      hashKey.getAttributeValue( "name" ), this.targetNamespace.getPrefix() );

      complexTypes.put( hashValue, hashKey );
   }

}   // End extractComplexTypes


/** ****************************************************************************
 * <br>
 * This function parses through the explicitly defined simpleTypes in the
 * schema and builds a discriminator vector for each one. The discriminator
 * vectors constructed by this function are then added to the simpleTypes
 * hash map along with the name of each simple type.
 *
 * @param None
```

```
 * @return None
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
protected void extractSimpleTypes()
{

   // Get a list of all explicitly declared elements in Schema.
   List simples = this.docRoot.getChildren( "simpleType", this.docNamespace );

   // Get iterator for simples list
   Iterator simpleIterator = simples.iterator();

   // iterate through the list of simple types
   while ( simpleIterator.hasNext() )
   {

      Element simpleNode = ( Element )simpleIterator.next();

      // Get the fully qualified name of the type defined in the element
      String attrNameValue = this.makeQualifiedElementName(
      simpleNode.getAttributeValue( "name" ), this.targetNamespace.getPrefix() );

      //create a discriminator vector
      DiscriminatorVector dv = buildSimpleVector( simpleNode );

      // Add the simple type and its discriminator vector to the simpleTypes
      // hash map.
      simpleTypes.put( attrNameValue, dv );

   }

}   // End extractSimpleTypes



/** ***************************************************************************
 * <br>
 * Builds a discriminator vector for a simpleType XML element.  The
 * parsing of restriction elements follows the W3C specifications for
 * built-in XML Schema primitive and derived data types.  These specifications
 * can be found at <a href="http://www.w3c.org/TR/xmlschema-2" target="_blank">
 * http://www.w3c.org/TR/xmlschema-2/</a>.  In general, a schema must conform
 * to these standards and contain
 *
 * @param simpleTypeNode - a simpleType element/node of an XML schema.
 * @return A discriminator describing the simpleTypeNode
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
protected DiscriminatorVector buildSimpleVector( Element simpleTypeNode )
{

   DiscriminatorVector result = new DiscriminatorVectorImpl();

   // Set the property type to indicate an atomic attribute. This value
   // does not get normalized.
   result.setDVPropertyType( 1.0 );

   //Get the name of the simpleType
   String typeName = simpleTypeNode.getAttributeValue( "name" );

   //Get the restriction element under the simpleType element
   Element restriction = simpleTypeNode.getChild( "restriction", this.docNamespace );

   String base = restriction.getAttributeValue( "base" );
```

```
Element length         = restriction.getChild( "length", docNamespace );
Element minLength      = restriction.getChild( "minLength", docNamespace );
Element maxLength      = restriction.getChild( "maxLength", docNamespace );
Element pattern        = restriction.getChild( "pattern", docNamespace );
List enumerations      = restriction.getChildren( "enumeration", docNamespace );
Element minInclusive   = restriction.getChild( "minInclusive", docNamespace );
Element maxInclusive   = restriction.getChild( "maxInclusive", docNamespace );
Element minExclusive   = restriction.getChild( "minExclusive", docNamespace );
Element maxExclusive   = restriction.getChild( "maxExclusive", docNamespace );
Element totalDigits    = restriction.getChild( "totalDigits", docNamespace );
Element fractionDigits = restriction.getChild( "fractionDigits", docNamespace );

if ( base.equals( "xsd:string" ) )
{

   // In accordance withe W3C XML Schema specification, the string primitive
   // data type can have the following constraining facets:
   //
   // length
   // minlength
   // maxlength
   // pattern
   // enumeration
   // whitespace
   //
   // The facet whitespace is not utilized in the discriminator vectors


   // Is there a length facet?
   if ( length != null )
   {
      // If yes, then set the min and max length properties
      // to the value of the length element
      double temp = Double.parseDouble( length.getAttributeValue( "value" ) );
      result.setDVMinLength( temp );
      result.setDVMaxLength( temp );
   }

   // Is there a minLength facet?
   if ( minLength != null )
   {
      // If yes, set the minLength restriction
      double temp = Double.parseDouble( minLength.getAttributeValue( "value" ) );
      result.setDVMinLength( temp );
   }

   // Is there a maxLength facet?
   if ( maxLength != null )
   {
      // If yes, set the maxLength restriction
      double temp = Double.parseDouble( maxLength.getAttributeValue( "value" ) );
      result.setDVMaxLength( temp );
   }

   // Is there a defined pattern facet?
   if ( pattern != null )
   {
      // If yes, set the pattern property to 1.0
      result.setDVPattern( 1.0 );
   }

   // Are there enumeration facets?
   if ( !enumerations.isEmpty() )
   {
      // If yes, get the number of enumerations and put in vector
      result.setDVNumEnumerations( ( double )enumerations.size() );
   }

   // Lastly, set the data type proporty to 1.0 to denote the data type
```

```
      result.setDVStringType( 1.0 );

   }
   else if ( base.equals( "xsd:boolean" ) )
   {

      // In accordance withe W3C XML Schema specification, the boolean primitive
      // data type can have the following constraining facets:
      //
      // pattern
      // whitespace
      //
      // The facet whitespace is not utilized in the discriminator vectors


      // Is there a defined pattern facet?
      if ( pattern != null )
      {
         // If yes, set the pattern property to 1.0
         result.setDVPattern( 1.0 );
      }

      // Lastly, set the data type property to 1.0 to denote the data type
      result.setDVBooleanType( 1.0 );

   }
   else if ( base.equals( "xsd:float" ) || base.equals( "xsd:double" ) )
   {

      // In accordance withe W3C XML Schema specification, the float and
      // double primitive data types can have the following constraining facets:
      //
      // pattern
      // enumeration
      // whitespace
      // minInclusive
      // maxInclusive
      // minExclusive
      // maxExclusive
      //
      // The facet whitespace is not utilized in the discriminator vectors


      // Is there a defined pattern facet?
      if ( pattern != null )
      {
         // If yes, set the pattern property to 1.0
         result.setDVPattern( 1.0 );
      }

      // Are there enumeration facets?
      if ( !enumerations.isEmpty() )
      {
         // If yes, get the number of enumerations and put in vector
         result.setDVNumEnumerations( ( double )enumerations.size() );
      }

      // Is there a minInclusive facet?
      if ( minInclusive != null )
      {
         // If yes, set the minInclusive restriction
         double temp = Double.parseDouble(minInclusive.getAttributeValue("value"));
         result.setDVMinInclusive( temp );
      }

      // Is there a maxInclusive facet?
      if ( maxInclusive != null )
      {
         // If yes, set the maxInclusive restriction
```

```
      double temp = Double.parseDouble(maxInclusive.getAttributeValue("value"));
      result.setDVMaxInclusive( temp );
   }

   // Is there a minExclusive facet?
   if ( minExclusive != null )
   {
      // If yes, set the minExclusive restriction
      double temp = Double.parseDouble(minExclusive.getAttributeValue("value"));
      result.setDVMinExclusive( temp );
   }

   // Is there a maxExclusive facet?
   if ( maxExclusive != null )
   {
      // If yes, set the maxExclusive restriction
      double temp = Double.parseDouble(maxExclusive.getAttributeValue("value"));
      result.setDVMaxExclusive( temp );
   }

   // Lastly, set the data type property to 1.0 to denote the data type
   if ( base.equals( "xsd:float" ) ) { result.setDVFloatType( 1.0 ); }
   if ( base.equals( "xsd:double" ) ) { result.setDVDoubleType( 1.0 ); }

}
else if ( base.equals( "xsd:BigDecimal" ) || base.equals( "xsd:integer" ) ||
base.equals( "xsd:long" ) || base.equals( "xsd:short" ) )
{

   // In accordance withe W3C XML Schema specification, the decimal
   // primitive data type can have the following constraining facets:
   //
   // totalDigits
   // fractionDigits
   // pattern
   // enumeration
   // whitespace
   // minInclusive
   // maxInclusive
   // minExclusive
   // maxExclusive
   //
   // The facet whitespace is not utilized in the discriminator vectors


   // Is there a totalDigits facet?
   if ( totalDigits != null )
   {
      // If yes, set the total digits property
      double temp = Double.parseDouble(totalDigits.getAttributeValue("value" ));
      result.setDVTotalDigits( temp );
   }

   // Is there a fractionDigits facet?
   if ( fractionDigits != null )
   {
      // If yes, set the fraction digits property
      double temp = Double.parseDouble(fractionDigits.getAttributeValue("value"));
      result.setDVFractionDigits( temp );
   }

   // Is there a defined pattern facet?
   if ( pattern != null )
   {
      // If yes, set the pattern property to 1.0
      result.setDVPattern( 1.0 );
   }

   // Are there enumeration facets?
```

```
      if ( !enumerations.isEmpty() )
      {
         // If yes, get the number of enumerations and put in vector
         result.setDVNumEnumerations( ( double )enumerations.size() );
      }

      // Is there a minInclusive facet?
      if ( minInclusive != null )
      {
         // If yes, set the minInclusive restriction
         double temp = Double.parseDouble(minInclusive.getAttributeValue("value"));
         result.setDVMinInclusive( temp );
      }

      // Is there a maxInclusive facet?
      if ( maxInclusive != null )
      {
         // If yes, set the maxInclusive restriction
         double temp = Double.parseDouble(maxInclusive.getAttributeValue("value"));
         result.setDVMaxInclusive( temp );
      }

      // Is there a minExclusive facet?
      if ( minExclusive != null )
      {
         // If yes, set the minExclusive restriction
         double temp = Double.parseDouble(minExclusive.getAttributeValue("value"));
         result.setDVMinExclusive( temp );
      }

      // Is there a maxExclusive facet?
      if ( maxExclusive != null )
      {
         // If yes, set the maxExclusive restriction
         double temp = Double.parseDouble(maxExclusive.getAttributeValue("value"));
         result.setDVMaxExclusive( temp );
      }

      // Set the data type property to 1.0 to denote the data type
      if ( base.equals( "xsd:BigDecimal" )
          ) { result.setDVBigDecimalType( 1.0 ); }
      if ( base.equals( "xsd:integer" ) ) { result.setDVIntType( 1.0 ); }
      if ( base.equals( "xsd:long" ) ) { result.setDVLongType( 1.0 ); }
      if ( base.equals( "xsd:short" ) ) { result.setDVShortType( 1.0 ); }

   }
   else
   {
      // Lastly, set the data type property to 1.0 to denote the data type
      result.setDVOtherType( 1.0 );

   }

   return result;

}   // End buildSimpleVector


/** ***************************************************************************
 * <br>
 * Recursive function to merge the discriminator vectors of the child data types
 * with the vectors of the parent types.  The merge method
 * traverses the ArrayList
 * passed in as param 1 and modifies the values of the
 * discriminator vector passed
 * by reference as param 2. (i.e) index 0 must be the name of data type, index 1
 * must be the data types discriminator vector, and index 2 begins the
 * children associated with the data type.
 *
```

```
 * @param parentList - An ArrayList conforming to the correlator schema.
 * @return None
 *
 * @author LT Steve Shedd
 *
 ************************************************************************** */
public void mergeVectors( ArrayList parentList ) throws Exception
{

   int parentListSize = parentList.size();

   DiscriminatorVector parentDV = ( DiscriminatorVector )parentList.get( 1 );

   if ( parentListSize > 2 )
   {

      for ( int z = 2; z < parentListSize; z++ )
      {

         ArrayList nextChildList = ( ArrayList )parentList.get( z );

         // Make recursive calls to the mergeVector routine.
         // As shown in the second parameter, it is crucial that the
         // discriminator vector be stored in the 2nd index of the list
         // which is index 1 in a 0-based indexing scheme.
         mergeVectors( nextChildList );

         DiscriminatorVector childDV = (DiscriminatorVector)nextChildList.get(1);

         /*
         System.out.print("\nMerging " + nextChildList.get(0) +
         " with parent " + parentList.get(0) + " after recursion:");

         float[] child = childDV.getValuesAsFloatArray();

         System.out.print("\nchild:\t");

         for ( int i = 0; i < child.length ; i++ ) {

            System.out.print( child[i] + " ");

         }

         float[] parent = parentDV.getValuesAsFloatArray();

         System.out.print("\nparent:\t");

         for ( int j = 0; j < parent.length ; j++ ) {

            System.out.print( parent[j] + " ");

         }
         */


         parentDV.mergeChildWithParent( childDV );


         /*
         float[] after = parentDV.getValuesAsFloatArray();

         System.out.print("\nresult:\t");

         for ( int k = 0; k < after.length ; k++ ) {

            System.out.print( after[k] + " ");

         }
         */
```

```
         }

      }

   }   // End mergeVectors


   /** ****************************************************************************
    * <br>
    * Small utility method to generate a fully qualified name of schema type. The
    * method simply concatenates the target namespace of the schema to the front of
    * the schema data type.
    *
    * @param elementName - The name of the data type
    * @param targetNamespace - The name of the targetNamespace
    * @return The qualified name of the schema type
    *
    * @author LT Steve Shedd
    *
    ******************************************************************************* */
   private String makeQualifiedElementName( String elementName,
   String targetNamespace )
   {
      return targetNamespace + ":" + elementName;
   }


}      // End DiscriminatorGenerator
```

3. Interface DiscriminatorVector

```
//*****************************************************************************
// Filename:.............DiscriminatorVector.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************


package mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator;

import java.lang.Exception;

/**
* The <i>DiscriminatorVector</i> interface defines the services required
* for an implementation of a discriminator vector class. The discriminator
* vector includes the logic for normalization and conversion to other data
* types.
*
* @author LT Steve Shedd
* @version 1.0 September, 2002
*
*/
public interface DiscriminatorVector
{

   /**
    * Normalizes the current raw values of the vector in
    * accrodance with the speicific normalization routine
    * defined.
    *
    * @throws Exception if there is a math error in the normalization routine
    * @author LT Steve Shedd
    */
```

```
public void normalizeValues() throws Exception;


/**
 * @return the discriminator vector as an array of doubles
 * @author LT Steve Shedd
 */
public double[] ValuesAsDoubleArray();

/**
 * @return the discriminator vector as an array of floats
 * @author LT Steve Shedd
 */
public float[] ValuesAsFloatArray();

/**
 * Utility function that combines two discriminator vectors in
 * accordance with predefined rules.
 *
 * @param in - A discriminator vector to merge with the current vector
 * @return None
 * @author LT Steve Shedd
 */
public void mergeChildWithParent( DiscriminatorVector in ) throws Exception;

/**
 * @return true if the vector describes an attribute, false otherwise.
 * @author LT Steve Shedd
 */
public boolean isAttribute();

/**
 * @return true if the vector is a complex attribute, false otherwise.
 * @author LT Steve Shedd
 */
public boolean isComplex();

/**
 * @return true if the vector is normalized, false otherwise.
 * @author LT Steve Shedd
 */
public boolean isNormalized();

/**
 * @return true if the vector describes an operation, false otherwise.
 * @author LT Steve Shedd
 */
public boolean isOperation();


/**
 * The get and set methods that follow are for discriminate properties
 * used in the discriminator vector.
 *
 * @author Steve Shedd
 */
public double getDVPropertyType();
public double getDVIsComplex();
public double getDVNumSubTypes();
public double getDVNumReqdSubTypes();
public double getDVNumOptSubTypes();
public double getDVNumOperations();
public double getDVNumParameters();
public double getDVStringType();
public double getDVBooleanType();
public double getDVFloatType();
public double getDVDoubleType();
public double getDVBigDecimalType();
public double getDVIntType();
```

```
   public double getDVLongType();
   public double getDVShortType();
   public double getDVOtherType();
   public double getDVMinOccurs();
   public double getDVMaxOccurs();
   public double getDVMinLength();
   public double getDVMaxLength();
   public double getDVTotalDigits();
   public double getDVFractionDigits();
   public double getDVPattern();
   public double getDVNumEnumerations();
   public double getDVMinExclusive();
   public double getDVMaxExclusive();
   public double getDVMinInclusive();
   public double getDVMaxInclusive();

   public void setDVPropertyType( double value );
   public void setDVIsComplex( double value );
   public void setDVNumSubTypes( double value );
   public void setDVNumReqdSubTypes( double value );
   public void setDVNumOptSubTypes( double value );
   public void setDVNumOperations( double value );
   public void setDVNumParameters( double value );
   public void setDVStringType( double value );
   public void setDVBooleanType( double value );
   public void setDVFloatType( double value );
   public void setDVDoubleType( double value );
   public void setDVBigDecimalType( double value );
   public void setDVIntType( double value );
   public void setDVLongType( double value );
   public void setDVShortType( double value );
   public void setDVOtherType( double value );
   public void setDVMinOccurs( double value );
   public void setDVMaxOccurs( double value );
   public void setDVMinLength( double value );
   public void setDVMaxLength( double value );
   public void setDVTotalDigits( double value );
   public void setDVFractionDigits( double value );
   public void setDVPattern( double value );
   public void setDVNumEnumerations( double value );
   public void setDVMinExclusive( double value );
   public void setDVMaxExclusive( double value );
   public void setDVMinInclusive( double value );
   public void setDVMaxInclusive( double value );


} // End DiscriminatorVector
```

4. Class DiscriminatorVectorImpl

```
//*****************************************************************************
// Filename:.............DiscriminatorVectorImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator;

import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.Iterator;

import java.lang.Double;
```

```
import java.lang.Math;

import java.io.IOException;


/** ****************************************************************************
 * The <i>DiscriminatorVectorImpl</i> class stores the values of a
 * discriminator vector.  It includes the logic for normalization and
 * conversion of the vectore to other data types.  The class is not used outside
 * of the syntacticComponentGeneration module.
 *
 * @author LT Steve Shedd
 * @version 1.0 September, 2002
 *
 **************************************************************************** */
public class DiscriminatorVectorImpl implements DiscriminatorVector
{


   /** ************************************************************************
    *  Data structure that holds the enumeration of the discrimintor vector
    *  elements and the values of those elements.  This data type was chosen
    *  becuase of its inherent ability to store a key and value.  In this case
    *  the key is a string representing the element label and the value is the
    *  actual value of the element represented as a double
    ************************************************************************** */
   private HashMap dv = null;


   /** ************************************************************************
    *  Data structure that holds a backup copy of the discriminator vector.  The
    *  purpose of this class attribute is for a future undo feature if desired.
    ************************************************************************** */
   private HashMap backup = null;


   /** ************************************************************************
    *  Boolean control variable to test whether the data contained in the
    *  discriminator vector has been normalized or not.
    ************************************************************************** */
   private boolean isNormalized = false;


   /** Boolean to track whether the vector is for an operation or not. */
   private boolean isOperation = false;


   /** Boolean to track whether the vector is a complex type or not. */
   private boolean isComplex = false;


   /** Boolean to track whether the vector is an atomic attribute or not. */
   private boolean isAttribute = true;


   /** Default no-arg constructor. */
   public DiscriminatorVectorImpl()
   {
      initVector();
   }


   /** ************************************************************************
    *  Copy constructor.  This constructor initializes a descriminator
    *  vector hash map with the values of another disciminator vector
    *  passed in as an argument.
    *
    *  @author LT Steve Shedd
    *  @param A discriminator vector from which another vector will be created.
```

```
 *
 ********************************************************************************* */
public DiscriminatorVectorImpl( DiscriminatorVector in )
{

   initVector();

   // Set local control booleans
   this.isOperation = in.isOperation();
   this.isAttribute = in.isAttribute();
   this.isComplex = in.isComplex();
   this.isNormalized = in.isNormalized();

   this.setDVPropertyType( in.getDVPropertyType() );
   this.setDVIsComplex( in.getDVIsComplex() );
   this.setDVNumSubTypes( in.getDVNumSubTypes() );
   this.setDVNumReqdSubTypes( in.getDVNumReqdSubTypes() );
   this.setDVNumOptSubTypes( in.getDVNumOptSubTypes() );
   this.setDVNumOperations( in.getDVNumOperations() );
   this.setDVNumParameters( in.getDVNumParameters() );
   this.setDVStringType( in.getDVStringType() );
   this.setDVBooleanType( in.getDVBooleanType() );
   this.setDVFloatType( in.getDVFloatType() );
   this.setDVDoubleType( in.getDVDoubleType() );
   this.setDVBigDecimalType( in.getDVBigDecimalType() );
   this.setDVIntType( in.getDVIntType() );
   this.setDVLongType( in.getDVLongType() );
   this.setDVShortType( in.getDVShortType() );
   this.setDVOtherType( in.getDVOtherType() );
   this.setDVMinOccurs( in.getDVMinOccurs() );
   this.setDVMaxOccurs( in.getDVMaxOccurs() );
   this.setDVMinLength( in.getDVMinLength() );
   this.setDVMaxLength( in.getDVMaxLength() );
   this.setDVTotalDigits( in.getDVTotalDigits() );
   this.setDVFractionDigits( in.getDVFractionDigits() );
   this.setDVPattern( in.getDVPattern() );
   this.setDVNumEnumerations( in.getDVNumEnumerations() );
   this.setDVMinExclusive( in.getDVMinExclusive() );
   this.setDVMaxExclusive( in.getDVMaxExclusive() );
   this.setDVMinInclusive( in.getDVMinInclusive() );
   this.setDVMaxInclusive( in.getDVMaxInclusive() );

   this.backup = new HashMap( this.dv );

}   // End copy constructor


/** ***************************************************************************
 *  Private utility method to initialize discriminator vector.  Called from
 *  within the constructors.
 *
 * @param None
 * @return None
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
private void initVector()
{

   // Create a new hash map with a capacity of 28 elements
   dv = new HashMap( 28 );

   // Initialize the hash map with a place holder foreach element
   // of a discriminator vector.
   dv.put( "propertyType", new Double( 1.0 ) );        // vector property 1
   dv.put( "isComplex", new Double( 0.0 ) );           // vector property 2
   dv.put( "numSubTypes", new Double( 0.0 ) );         // vector property 3
   dv.put( "numReqdSubTypes", new Double( 0.0 ) );     // vector property 4
```

```
   dv.put( "numOptSubTypes", new Double( 0.0 ) );      // vector property 5
   dv.put( "numOperations", new Double( 0.0 ) );       // vector property 6
   dv.put( "numParameters", new Double( 0.0 ) );       // vector property 7
   dv.put( "stringType", new Double( 0.0 ) );          // vector property 8
   dv.put( "booleanType", new Double( 0.0 ) );         // vector property 9
   dv.put( "floatType", new Double( 0.0 ) );           // vector property 10
   dv.put( "doubleType", new Double( 0.0 ) );          // vector property 11
   dv.put( "bigDecimalType", new Double( 0.0 ) );      // vector property 12
   dv.put( "intType", new Double( 0.0 ) );             // vector property 13
   dv.put( "longType", new Double( 0.0 ) );            // vector property 14
   dv.put( "shortType", new Double( 0.0 ) );           // vector property 15
   dv.put( "otherType", new Double( 0.0 ) );           // vector property 16
   dv.put( "minOccurs", new Double( 0.0 ) );           // vector property 17
   dv.put( "maxOccurs", new Double( 0.0 ) );           // vector property 18
   dv.put( "minLength", new Double( 0.0 ) );           // vector property 19
   dv.put( "maxLength", new Double( 0.0 ) );           // vector property 20
   dv.put( "totalDigits", new Double( 0.0 ) );         // vector property 21
   dv.put( "fractionDigits", new Double( 0.0 ) );      // vector property 22
   dv.put( "pattern", new Double( 0.0 ) );             // vector property 23
   dv.put( "numEnumerations", new Double( 0.0 ) );     // vector property 24
   dv.put( "minExclusive", new Double( 0.0 ) );        // vector property 25
   dv.put( "maxExclusive", new Double( 0.0 ) );        // vector property 26
   dv.put( "minInclusive", new Double( 0.0 ) );        // vector property 27
   dv.put( "maxInclusive", new Double( 0.0 ) );        // vector property 28

}   // End initVector()


////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                    Vector Normalization Section                            //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////

/** ***************************************************************************
 *   The major utility function of the DiscriminatorVector class.  This function
 *   takes an instance of a DiscriminatorVector and normalizes each value of each
 *   element in accrodance with the OOMI correlation normalization schemes.
 *
 *   The neural network used for the correlation process requires each value of
 *   an element to be in the range [0.0,1.0].
 *
 *   @author  LT Steve Shedd
 *   @return  A normalized instance of the DiscriminatorVector with
 *            element values in the range [0.0,1.0]
 **************************************************************************** */
public void normalizeValues() throws Exception
{

   if ( this.isNormalized() )
   {
      throw new Exception( "Cannot normalize a vector that has already" +
      " already been normalized." );
   }
   else
   {

      // create a backup copy of the hashmap.  This feature is included for a possible
      // undo feature if desired in the futre.
      this.backup = new HashMap( this.dv );

      // Set local control booleans
      boolean operation = getDVPropertyType() == 0.0 ? true : false;
      boolean attribute = getDVPropertyType() == 1.0 ? true : false;
      boolean complex   = getDVIsComplex() == 1.0 ? true : false;

      // Normalize each value of the discriminator vector.
      //
      // We do not normalize the following properties:
```

```
      // propertyType
      // isComplex
      // minOccurs
      //

      if ( complex )
      {
         setDVNumSubTypes( normalizeTwo( getDVNumSubTypes() ) );
         setDVNumReqdSubTypes( normalizeTwo( getDVNumReqdSubTypes() ) );
         setDVNumOptSubTypes( normalizeTwo( getDVNumOptSubTypes() ) );
         setDVNumOperations( normalizeTwo( getDVNumOperations() ) );
         setDVPattern( normalizeTwo( getDVPattern() ) );
      }

      if ( complex || operation )
      {
         setDVNumParameters( normalizeTwo( getDVNumParameters() ) );
         setDVStringType( normalizeTwo( getDVStringType() ) );
         setDVBooleanType( normalizeTwo( getDVBooleanType() ) );
         setDVFloatType( normalizeTwo( getDVFloatType() ) );
         setDVDoubleType( normalizeTwo( getDVDoubleType() ) );
         setDVBigDecimalType( normalizeTwo( getDVBigDecimalType() ) );
         setDVIntType( normalizeTwo( getDVIntType() ) );
         setDVLongType( normalizeTwo( getDVLongType() ) );
         setDVShortType( normalizeTwo( getDVShortType() ) );
         setDVOtherType( normalizeTwo( getDVOtherType() ) );
      }

      if ( !operation )
      {
         setDVMaxOccurs( normalizeTwo( getDVMaxOccurs() ) );
         setDVMinLength( normalizeTwo( getDVMinLength() ) );
         setDVMaxLength( normalizeThree( getDVMaxLength() ) );
         setDVTotalDigits( normalizeThree( getDVTotalDigits() ) );

         //if ( this.getDVFractionDigits() == 0.0 ) {
         this.setDVFractionDigits( 0.0 );
         //}
         //else {
         // this.setDVFractionDigits( normalizeThree( getDVFractionDigits() ) );
         //}

         if ( this.getDVNumEnumerations() == 0.0 )
         {
            this.setDVNumEnumerations( 0.0 );
         }
         else
         {
            this.setDVNumEnumerations( normalizeTwo( getDVNumEnumerations() ) );
         }

         setDVMinExclusive( normalizeThree( getDVMinExclusive() ) );
         setDVMaxExclusive( normalizeThree( getDVMaxExclusive() ) );
         setDVMinInclusive( normalizeThree( getDVMinInclusive() ) );
         setDVMaxInclusive( normalizeThree( getDVMaxInclusive() ) );

      }

      // Set the normalized boolean to true.
      this.isNormalized = true;

   }

}   // End normalizeValues


/** ************************************************************************
 * The first noralization function defined in the methodology.
 *
```

```
 * @param x - a double value to be normalized
 * @return the normalization of the input
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
protected double normalizeOne( double x )
{
   double k = 1.01;
   return  1 / ( 1 + Math.pow( k, -x ) );
}


/** The reverse normalization of normOne if ever needed */
protected int reverseNormOne( double y )
{
   double k   = 1.01;
   Double x   = new Double( Math.log( y / ( 1 - y ) ) / Math.log( k ) );
   int result = x.intValue();
   return result;
}


/** ********************************************************************************
 * The second noralization function defined in the methodology
 *
 * @param x - a double value to be normalized
 * @return the normalization of the input
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
protected double normalizeTwo( double x )
{
   double k = 1.01;
   return  2 * ( 1 / ( 1 + Math.pow( k, -x ) ) ) - 0.5 ;
}


/** ********************************************************************************
 * The third noralization functions defined in the methodology
 *
 * @param x - a double value to be normalized
 * @return the normalization of the input
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
protected double normalizeThree( double x )
{
   return ( Math.log( x + 1 ) / Math.log( 10 ) ) / 5;
}



//////////////////////////////////////////////////////////////////////////////////
//                                                                              //
//             Discriminator Element Get/Set Methods Section                    //
//                                                                              //
//////////////////////////////////////////////////////////////////////////////////


/** ********************************************************************************
 *  Get the value of the private member variable isOperation.
 *
 *  @return  Return true if the discriminator vector describes an
 *           operation, false otherwise.
 *
 * @author LT Steve Shedd
```

```
 *
 ********************************************************************************* */
public boolean isOperation()
{
   return getDVPropertyType() == 0.0 ? true : false;
}


/** ********************************************************************************
 *  Get the value of the private member variable isComplex.
 *
 *  @return  Return true if the discriminator vector describes a
 *           complex attribute/type, false otherwise.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
public boolean isComplex()
{
   return getDVIsComplex() == 1.0 ? true : false;
}


/** ********************************************************************************
 *  Get the value of the private member variable isAttribute.
 *
 *  @return  Return true if the discriminator vector describes an
 *           atomic attribute, false otherwise.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
public boolean isAttribute()
{
   return getDVPropertyType() == 1.0 ? true : false;
}


/** ********************************************************************************
 *  Get the value of the private member variable isNormalized.
 *
 *  @return  Return true if the values of the discriminator vector have
 *           been normalized to double in the range [0.0, 1.0], false otherwise.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
public boolean isNormalized()
{
   return this.isNormalized;
}


//*****     Discriminate #1    *****//

/** ********************************************************************************
 *  Get the value of the discriminator vector element  property type
 *
 *  @return  The value of the proporty type element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
public double getDVPropertyType()
{
   Double value = ( Double )dv.get( "propertyType" );
   return value.doubleValue();
}
```

```
/** Set the value of the discriminator vector element  property type */
public void setDVPropertyType( double value )
{
   this.dv.put( "propertyType", new Double( value ) );

   if ( value == 1.0 )
   {
      this.isAttribute = true;
      this.isOperation = false;
   }
   else
   {
      this.isAttribute = false;
      this.isOperation = true;
   }

}


//*****     Discriminate #2    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  isComplex
 *
 *  @return  The value of the isComplex element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVIsComplex()
{
   Double value = ( Double )dv.get( "isComplex" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  isComplex */
public void setDVIsComplex( double value )
{
   this.dv.put( "isComplex", new Double( value ) );

   if ( value == 1.0 )
   {
      this.isComplex = true;
   }
   else
   {
      this.isComplex = false;
   }

}


//*****    Discriminate #3    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  numSubTypes
 *
 *  @return  The value of the numSubTypes element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVNumSubTypes()
{
   Double value = ( Double )dv.get( "numSubTypes" );
   return value.doubleValue();
}
```

```
/** Set the value of the discriminator vector element  numSubTypes */
public void setDVNumSubTypes( double value )
{
   this.dv.put( "numSubTypes", new Double( value ) );
}


//*****    Discriminate  #4    *****//


/** ****************************************************************************
 *  Get the value of the discriminator vector element  numReqdSubTypes
 *
 *  @return  The value of the numReqdSubTypes element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVNumReqdSubTypes()
{
   Double value = ( Double )dv.get( "numReqdSubTypes" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  numReqdSubTypes */
public void setDVNumReqdSubTypes( double value )
{
   this.dv.put( "numReqdSubTypes", new Double( value ) );
}


//*****     Discriminate #5    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  numOptSubTypes
 *
 *  @return  The value of the numOptSubTypes element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVNumOptSubTypes()
{
   Double value = ( Double )dv.get( "numOptSubTypes" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  numOptSubTypes */
public void setDVNumOptSubTypes( double value )
{
   this.dv.put( "numOptSubTypes", new Double( value ) );
}


//*****     Discriminate #6    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  numOperations
 *
 *  @return  The value of the numOperations element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVNumOperations()
```

```
{
   Double value = ( Double )dv.get( "numOperations" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  numOperations */
public void setDVNumOperations( double value )
{
   this.dv.put( "numOperations", new Double( value ) );
}


//*****     Discriminate #7    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  numParameters
 *
 *  @return  The value of the numParameters element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVNumParameters()
{
   Double value = ( Double )dv.get( "numParameters" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  numParameters */
public void setDVNumParameters( double value )
{
   this.dv.put( "numParameters", new Double( value ) );
}


//*****     Discriminate #8    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  stringType
 *
 *  @return  The value of the stringType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVStringType()
{
   Double value = ( Double )dv.get( "stringType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  stringType */
public void setDVStringType( double value )
{
   this.dv.put( "stringType", new Double( value ) );
}


//*****     Discriminate #9    *****//


/** ****************************************************************************
 *  Get the value of the discriminator vector element  booleanType
 *
 *  @return  The value of the booleanType element of the discriminator vector
 *
```

```
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public double getDVBooleanType()
{
   Double value = ( Double )dv.get( "booleanType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  booleanType */
public void setDVBooleanType( double value )
{
   this.dv.put( "booleanType", new Double( value ) );
}


//*****     Discriminate #10    *****//

/** ********************************************************************************
 *  Get the value of the discriminator vector element  floatType
 *
 *  @return  The value of the floatType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public double getDVFloatType()
{
   Double value = ( Double )dv.get( "floatType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  floatType */
public void setDVFloatType( double value )
{
   this.dv.put( "floatType", new Double( value ) );
}


//*****     Discriminate #11    *****//

/** ********************************************************************************
 *  Get the value of the discriminator vector element  doubleType
 *
 *  @return  The value of the doubleType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public double getDVDoubleType()
{
   Double value = ( Double )dv.get( "doubleType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  doubleType */
public void setDVDoubleType( double value )
{
   this.dv.put( "doubleType", new Double( value ) );
}


//*****     Discriminate #12    *****//


/** ********************************************************************************
```

```
 *  Get the value of the discriminator vector element  bigDecimalType
 *
 *  @return  The value of the bigDecimalType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public double getDVBigDecimalType()
{
   Double value = ( Double )dv.get( "bigDecimalType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  bigDecimalType */
public void setDVBigDecimalType( double value )
{
   this.dv.put( "bigDecimalType", new Double( value ) );
}


//*****     Discriminate #13    *****//

/** ********************************************************************************
 *  Get the value of the discriminator vector element  intType
 *
 *  @return  The value of the intType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public double getDVIntType()
{
   Double value = ( Double )dv.get( "intType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  intType */
public void setDVIntType( double value )
{
   this.dv.put( "intType", new Double( value ) );
}


//*****    Discriminate  #14    *****//

/** ********************************************************************************
 *  Get the value of the discriminator vector element  longType
 *
 *  @return  The value of the longType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public double getDVLongType()
{
   Double value = ( Double )dv.get( "longType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  longType */
public void setDVLongType( double value )
{
   this.dv.put( "longType", new Double( value ) );
}
```

```
//*****     Discriminate #15    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  shortType
 *
 *  @return  The value of the shortType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVShortType()
{
   Double value = ( Double )dv.get( "shortType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  shortType */
public void setDVShortType( double value )
{
   this.dv.put( "shortType", new Double( value ) );
}


//*****     Discriminate #16    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  otherType
 *
 *  @return  The value of the otherType element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVOtherType()
{
   Double value = ( Double )dv.get( "otherType" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  otherType */
public void setDVOtherType( double value )
{
   this.dv.put( "otherType", new Double( value ) );
}


//*****     Discriminate #17    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  minOccurs
 *
 *  @return  The value of the minOccurs element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMinOccurs()
{
   Double value = ( Double )dv.get( "minOccurs" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  minOccurs */
public void setDVMinOccurs( double value )
{
   this.dv.put( "minOccurs", new Double( value ) );
```

```
}


//*****     Discriminate #18    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  maxOccurs
 *
 *  @return  The value of the maxOccurs element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMaxOccurs()
{
   Double value = ( Double )dv.get( "maxOccurs" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  maxOccurs */
public void setDVMaxOccurs( double value )
{
   this.dv.put( "maxOccurs", new Double( value ) );
}


//*****     Discriminate #19    *****//


/** ****************************************************************************
 *  Get the value of the discriminator vector element  minLength
 *
 *  @return  The value of the minLength element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMinLength()
{
   Double value = ( Double )dv.get( "minLength" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  minLength */
public void setDVMinLength( double value )
{
   this.dv.put( "minLength", new Double( value ) );
}


//*****    Discriminate  #20    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  maxLength
 *
 *  @return  The value of the maxLength element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMaxLength()
{
   Double value = ( Double )dv.get( "maxLength" );
   return value.doubleValue();
}
```

```
/** Set the value of the discriminator vector element  maxLength */
public void setDVMaxLength( double value )
{
   this.dv.put( "maxLength", new Double( value ) );
}


//*****     Discriminate #21    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  totalDigits
 *
 *  @return  The value of the totalDigits element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVTotalDigits()
{
   Double value = ( Double )dv.get( "totalDigits" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  totalDigits */
public void setDVTotalDigits( double value )
{
   this.dv.put( "totalDigits", new Double( value ) );
}


//*****     Discriminate #22    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  fractionDigits
 *
 *  @return  The value of the fractionDigits element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVFractionDigits()
{
   Double value = ( Double )dv.get( "fractionDigits" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  fractionDigits */
public void setDVFractionDigits( double value )
{
   this.dv.put( "fractionDigits", new Double( value ) );
}


//*****    Discriminate  #23    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  pattern
 *
 *  @return  The value of the pattern element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVPattern()
{
   Double value = ( Double )dv.get( "pattern" );
   return value.doubleValue();
```

```
}


/** Set the value of the discriminator vector element  pattern */
public void setDVPattern( double value )
{
   this.dv.put( "pattern", new Double( value ) );
}


//*****    Discriminate  #24    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  numEnumerations
 *
 *  @return  The value of the numEnumerations element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVNumEnumerations()
{
   Double value = ( Double )dv.get( "numEnumerations" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  numEnumerations */
public void setDVNumEnumerations( double value )
{
   this.dv.put( "numEnumerations", new Double( value ) );
}


//*****     Discriminate #25    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  minExclusive
 *
 *  @return  The value of the minExclusive element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMinExclusive()
{
   Double value = ( Double )dv.get( "minExclusive" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  minExclusive */
public void setDVMinExclusive( double value )
{
   this.dv.put( "minExclusive", new Double( value ) );
}


//*****    Discriminate #26    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  maxExclusive
 *
 *  @return  The value of the maxExclusive element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMaxExclusive()
```

```
{
   Double value = ( Double )dv.get( "maxExclusive" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  maxExclusive */
public void setDVMaxExclusive( double value )
{
   this.dv.put( "maxExclusive", new Double( value ) );
}


//*****   Discriminate  #27    *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  minInclusive
 *
 *  @return  The value of the minInclusive element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMinInclusive()
{
   Double value = ( Double )dv.get( "minInclusive" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  minInclusive */
public void setDVMinInclusive( double value )
{
   this.dv.put( "minInclusive", new Double( value ) );
}


//*****     Discriminate #28   *****//

/** ****************************************************************************
 *  Get the value of the discriminator vector element  maxInclusive
 *
 *  @return  The value of the maxInclusive element of the discriminator vector
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public double getDVMaxInclusive()
{
   Double value = ( Double )dv.get( "maxInclusive" );
   return value.doubleValue();
}


/** Set the value of the discriminator vector element  maxInclusive */
public void setDVMaxInclusive( double value )
{
   this.dv.put( "maxInclusive", new Double( value ) );
}


////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                       Utility Functions Section                        //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

/** ****************************************************************************
 * Converts the values of the Discriminator vector elements into a float array.
```

```
 * The ordering of the array elements is critical and conforms to the schema
 * outlined in the Correlator module.
 *
 * @return The discriminator vector as an array of floats
 *
 * @author LT Steve Shedd
 *
 ************************************************************************** */
public float[] ValuesAsFloatArray()
{
   float[] vector = new float[28];

   vector[0]  = ((Double)dv.get( "propertyType" ) ).floatValue();    // property 1
   vector[1]  = ((Double)dv.get( "isComplex" ) ).floatValue();       // property 2
   vector[2]  = ((Double)dv.get( "numSubTypes" ) ).floatValue();     // property 3
   vector[3]  = ((Double)dv.get( "numReqdSubTypes" ) ).floatValue(); // property 4
   vector[4]  = ((Double)dv.get( "numOptSubTypes" ) ).floatValue();  // property 5
   vector[5]  = ((Double)dv.get( "numOperations" ) ).floatValue();   // property 6
   vector[6]  = ((Double)dv.get( "numParameters" ) ).floatValue();   // property 7
   vector[7]  = ((Double)dv.get( "stringType" ) ).floatValue();      // property 8
   vector[8]  = ((Double)dv.get( "booleanType" ) ).floatValue();     // property 9
   vector[9]  = ((Double)dv.get( "floatType" ) ).floatValue();       // property 10
   vector[10] = ((Double)dv.get( "doubleType" ) ).floatValue();      // property 11
   vector[11] = ((Double)dv.get( "bigDecimalType" ) ).floatValue();  // property 12
   vector[12] = ((Double)dv.get( "intType" ) ).floatValue();         // property 13
   vector[13] = ((Double)dv.get( "longType" ) ).floatValue();        // property 14
   vector[14] = ((Double)dv.get( "shortType" ) ).floatValue();       // property 15
   vector[15] = ((Double)dv.get( "otherType" ) ).floatValue();       // property 16
   vector[16] = ((Double)dv.get( "minOccurs" ) ).floatValue();       // property 17
   vector[17] = ((Double)dv.get( "maxOccurs" ) ).floatValue();       // property 18
   vector[18] = ((Double)dv.get( "minLength" ) ).floatValue();       // property 19
   vector[19] = ((Double)dv.get( "maxLength" ) ).floatValue();       // property 20
   vector[20] = ((Double)dv.get( "totalDigits" ) ).floatValue();     // property 21
   vector[21] = ((Double)dv.get( "fractionDigits" ) ).floatValue();  // property 22
   vector[22] = ((Double)dv.get( "pattern" ) ).floatValue();         // property 23
   vector[23] = ((Double)dv.get( "numEnumerations" ) ).floatValue(); // property 24
   vector[24] = ((Double)dv.get( "minExclusive" ) ).floatValue();    // property 25
   vector[25] = ((Double)dv.get( "maxExclusive" ) ).floatValue();    // property 26
   vector[26] = ((Double)dv.get( "minInclusive" ) ).floatValue();    // property 27
   vector[27] = ((Double)dv.get( "maxInclusive" ) ).floatValue();    // property 28

   return vector;

} // End ValuesAsFloatArray


/** ************************************************************************
 * Converts the values of the Discriminator vector elements into a double array.
 * The ordering of the array elements is critical and conforms to the schema
 * outlined in the Correlator module.
 *
 * @return The discriminator vector as an array of doubles
 *
 * @author LT Steve Shedd
 *
 ************************************************************************** */
public double[] ValuesAsDoubleArray()
{
   double[] vector = new double[28];

   vector[0]  = ((Double)dv.get("propertyType" ) ).doubleValue();    // property 1
   vector[1]  = ((Double)dv.get("isComplex" ) ).doubleValue();       // property 2
   vector[2]  = ((Double)dv.get("numSubTypes" ) ).doubleValue();     // property 3
   vector[3]  = ((Double)dv.get("numReqdSubTypes" ) ).doubleValue(); // property 4
   vector[4]  = ((Double)dv.get("numOptSubTypes" ) ).doubleValue();  // property 5
   vector[5]  = ((Double)dv.get("numOperations" ) ).doubleValue();   // property 6
   vector[6]  = ((Double)dv.get("numParameters" ) ).doubleValue();   // property 7
   vector[7]  = ((Double)dv.get("stringType" ) ).doubleValue();      // property 8
   vector[8]  = ((Double)dv.get("booleanType" ) ).doubleValue();     // property 9
```

```
   vector[9]  = ((Double)dv.get("floatType" ) ).doubleValue();       // property 10
   vector[10] = ((Double)dv.get("doubleType" ) ).doubleValue();      // property 11
   vector[11] = ((Double)dv.get("bigDecimalType" ) ).doubleValue();  // property 12
   vector[12] = ((Double)dv.get("intType" ) ).doubleValue();         // property 13
   vector[13] = ((Double)dv.get("longType" ) ).doubleValue();        // property 14
   vector[14] = ((Double)dv.get("shortType" ) ).doubleValue();       // property 15
   vector[15] = ((Double)dv.get("otherType" ) ).doubleValue();       // property 16
   vector[16] = ((Double)dv.get("minOccurs" ) ).doubleValue();       // property 17
   vector[17] = ((Double)dv.get("maxOccurs" ) ).doubleValue();       // property 18
   vector[18] = ((Double)dv.get("minLength" ) ).doubleValue();       // property 19
   vector[19] = ((Double)dv.get("maxLength" ) ).doubleValue();       // property 20
   vector[20] = ((Double)dv.get("totalDigits" ) ).doubleValue();     // property 21
   vector[21] = ((Double)dv.get("fractionDigits" ) ).doubleValue();  // property 22
   vector[22] = ((Double)dv.get("pattern" ) ).doubleValue();         // property 23
   vector[23] = ((Double)dv.get("numEnumerations" ) ).doubleValue(); // property 24
   vector[24] = ((Double)dv.get("minExclusive" ) ).doubleValue();    // property 25
   vector[25] = ((Double)dv.get("maxExclusive" ) ).doubleValue();    // property 26
   vector[26] = ((Double)dv.get("minInclusive" ) ).doubleValue();    // property 27
   vector[27] = ((Double)dv.get("maxInclusive" ) ).doubleValue();    // property 28

   return vector;

}  // End ValuesAsDoubleArray


/** ************************************************************************
 * Merges the unnormalized values of a child discriminator vector with the
 * parent discriminator vector whos merge method is being invoked.  This
 * method is specifically designed to aid in the dicriminator vector
 * summations for complex types.
 *
 * @param A raw, unnormalized discriminator vector that is considered to
 * be a child of the discriminator vector invoking the merge method.
 * @return None
 * @throws Exception - if there is a general error in the merge process
 *
 * @author  LT Steve Shedd
 *
 ************************************************************************** */
public void mergeChildWithParent( DiscriminatorVector child ) throws Exception
{
   // Check to see of either vector has been normalized already.  If so,
   // then cannot add together.
   if ( this.isNormalized() || child.isNormalized() )
   {
      throw new Exception( "Cannot merge discriminator vectors that" +
      "are already normalized." );
   }
   else
   {
      double[] temp = new double[28];

      // This covers the cases of attr + attr, attr + oper, oper + attr,
      // and oper + oper.
      if ( this.getDVPropertyType() == 1.0 || child.getDVPropertyType() == 1.0 )
      {
         temp[0] = 1.0;
      }
      else
      {
         temp[0] = 0.0;
      }

      // Merging two discriminator vectors automatically results in a complex
      // vector, or a value of 1.0 in our schema.
      temp[1] = 1.0;

      // The parent vector must include the number of subtypes under the parent.
      // If the child has no associated subtypes, add 1 to the parent to
```

```
// count the child as a subtype.  Else, add the number of subtypes of the
// child to the subtypes of the parent plus 1.0 to account for the child
// itself.
if ( child.getDVNumSubTypes() == 0.0 )
{
   temp[2] = this.getDVNumSubTypes() + 1.0;
}
else
{
   temp[2] = this.getDVNumSubTypes() + child.getDVNumSubTypes() + 1.0;
}


// A type is required only if the minOccurs discriminant equals is
// equal to or greater than 1.0.  If the child has no tallies for subtypes
// and the child's minOccurs is 1.0 or greater, then add the value of the
// child minOccurs to the parent. Else, add 0.0 to the parent.
//if ( child.getDVNumReqdSubTypes() == 0.0 && child.getDVMinOccurs() >= 1.0 ) {
if ( child.getDVMinOccurs() >= 1.0 )
{
   temp[3] = this.getDVNumReqdSubTypes() + child.getDVNumReqdSubTypes() + 1.0;
}
else
{
   temp[3] = this.getDVNumReqdSubTypes() + child.getDVNumReqdSubTypes();
}


// A type is optional only if the minOccurs discriminant equals 0.0.
// If the child minOccurs is equal to 0.0 and the child has no tallies
// for subtypes, then add 1.0 to the parent.  Else, add the value of the
// child NumOptSubTypes to the parent.
//if ( child.getDVNumOptSubTypes() == 0.0 && child.getDVMinOccurs() == 0.0 ) {
if ( child.getDVMinOccurs() == 0.0 )
{
   temp[4] = this.getDVNumOptSubTypes() + child.getDVNumOptSubTypes() + 1.0;
}
else
{
   temp[4] = this.getDVNumOptSubTypes() + child.getDVNumOptSubTypes();
}


// A discriminator vector describes an operation if its property type
// equals 0.0.  If the child's propertyType equals 0.0 and the child has
// no tallies for subtypes, i.e. numOperations equals 0.0, then add 1.0
// to the parent. Else, add the child value to the parent.
if ( this.getDVPropertyType() == 0.0 && this.getDVNumOperations() == 0.0 )
{
   temp[5] = this.getDVNumOperations() + 1.0;
}
else
{
   temp[5] = this.getDVNumOperations() + child.getDVNumOperations();
}

// simple addition for the following
temp[6] = this.getDVNumParameters() + child.getDVNumParameters();
temp[7] = this.getDVStringType() + child.getDVStringType();
temp[8] = this.getDVBooleanType() + child.getDVBooleanType();
temp[9] = this.getDVFloatType() + child.getDVFloatType();
temp[10] = this.getDVDoubleType() + child.getDVDoubleType();
temp[11] = this.getDVBigDecimalType() + child.getDVBigDecimalType();
temp[12] = this.getDVIntType() + child.getDVIntType();
temp[13] = this.getDVLongType() + child.getDVLongType();
temp[14] = this.getDVShortType() + child.getDVShortType();
temp[15] = this.getDVOtherType() + child.getDVOtherType();

// If both vectors are optional, the resultant is optional.  If either
```

```
         // or both are mandatory, then the resultant is mandatory.
         if ( this.getDVMinOccurs() >= 1.0 || child.getDVMinOccurs() >= 1.0 )
         {
            temp[16] = 1.0;
         }
         else
         {
            temp[16] = 0.0;
         }

         // If the resultant vector is an attribute
         if ( temp[0] == 1.0 )
         {
            double x = this.getDVMaxOccurs();
            double y = child.getDVMaxOccurs();

            temp[17] =  x + y;
            temp[18] = this.getDVMinLength() + child.getDVMinLength();
            temp[19] = this.getDVMaxLength() + child.getDVMaxLength();
            temp[20] = this.getDVTotalDigits() + child.getDVTotalDigits();
            temp[21] = this.getDVFractionDigits() + child.getDVFractionDigits();
            temp[22] = this.getDVPattern() + child.getDVPattern();
            temp[23] = this.getDVNumEnumerations() + child.getDVNumEnumerations();
            temp[24] = this.getDVMinExclusive() + child.getDVMinExclusive();
            temp[25] = this.getDVMaxExclusive() + child.getDVMaxExclusive();
            temp[26] = this.getDVMinInclusive() + child.getDVMinInclusive();
            temp[27] = this.getDVMaxInclusive() + child.getDVMaxInclusive();
         }

         // Perform addition of values and insert into new discriminator vector.
         this.setDVPropertyType( temp[0] );
         this.setDVIsComplex( temp[1] );
         this.setDVNumSubTypes( temp[2] );
         this.setDVNumReqdSubTypes( temp[3] );
         this.setDVNumOptSubTypes( temp[4] );
         this.setDVNumOperations( temp[5] );
         this.setDVNumParameters( temp[6] );
         this.setDVStringType( temp[7] );
         this.setDVBooleanType( temp[8] );
         this.setDVFloatType( temp[9] );
         this.setDVDoubleType( temp[10] );
         this.setDVBigDecimalType( temp[11] );
         this.setDVIntType( temp[12] );
         this.setDVLongType( temp[13] );
         this.setDVShortType( temp[14] );
         this.setDVOtherType( temp[15] );
         this.setDVMinOccurs( temp[16] );
         this.setDVMaxOccurs( temp[17] );
         this.setDVMinLength( temp[18] );
         this.setDVMaxLength( temp[19] );
         this.setDVTotalDigits( temp[20] );
         this.setDVFractionDigits( temp[21] );
         this.setDVPattern( temp[22] );
         this.setDVNumEnumerations( temp[23] );
         this.setDVMinExclusive( temp[24] );
         this.setDVMaxExclusive( temp[25] );
         this.setDVMinInclusive( temp[26] );
         this.setDVMaxInclusive( temp[27] );

      }

   } // End mergeChildWithParent

}
```

5. Interface NeuralNetGenerator

```
//*****************************************************************************
// Filename:.............NeuralNetGenerator.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************


package mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator;

import java.util.List;

import mwa.ai.neural.Neural;
import mwa.ai.neural.NNfile;


/** ***************************************************************************
 * <br>
 * The <i>NeuralNetGenerator</i> interface defines the services that must be
 * provided by the a neural network generator implementation to the component model
 * correlator.  The class <i>ComponentModelCorrelator</i> uses this interface to
 * perform the creation of trained neural networks.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 *************************************************************************** */
public interface NeuralNetGenerator
{

   /** ************************************************************************
    * <br>
    * Returns a trained neural network for the list of discriminator vectors
    * passed as the parameter.
    *
    * @param discriminators - the list of discriminator vectors needed to
    * build the neural net.
    * @return a trained neural network.
    *
    * @author LT Steve Shedd
    *
    ************************************************************************** */
   public Neural generateNeuralNet( List discriminators );


   /** ************************************************************************
    * <br>
    * Makes a formatted file to represent the trained neural network.
    *  Returns true if the
    * file io was successful, false otherwise.
    *
    * @param nn - a trained neural network.
    * @param path - the path to be used to save the neural network file to disk.
    * @return true if the file was created and saved, false otherwise.
    *
    * @author LT Steve Shedd
    *
    ************************************************************************** */
   public boolean makeFile( Neural nn, String path );


}      // End NeuralNetGenerator
```

6. Class NeuralNetGeneratorImpl

```
//*****************************************************************************
// Filename:.............NeuralNetGeneratorImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************


package mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Vector;

import mwa.ai.neural.Neural;
import mwa.ai.neural.NNfile;


/** ***************************************************************************
 * <br>
 * The <i>NeuralNetGeneratorImpl</i> performs the generation and training
 * of neural networks
 * for use in the syntactic correlation process.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 **************************************************************************** */
public class NeuralNetGeneratorImpl implements NeuralNetGenerator
{

   /** number of input layer nodes for the neural network */
   static final int NUM_INPUT_NODES = 28;


   /** number of hidden layer nodes for the neural network */
   private int numHiddenNodes = 0;


   /** number of output layer nodes for the neural network */
   private int numOutputNodes = 0;


   /** Neural network learning rate */
   private float neuralLearningRate = 0.25f;


   /** Max number of Epochs allowed during neural network training */
   private int neuralMaxEpochs = 100000;


   /** Output vector error tolerance for training the neural network */
   private double neuralOutputErrorTolerance = 0.1;


   /** Default Constructor */
   public NeuralNetGeneratorImpl()
   {

   }


   /** ***************************************************************************
```

```
 * <br>
 * Returns a trained neural network for the list of discriminator vectors
 * passed as the parameter. The implementation provided below is convoluted due
 * to the use of the mwa.ai.neural package. In order to optimize the code, the
 * mwa.ai.neural package should be completely rewitten.
 *
 * @param discriminators - the list of discriminator vectors needed to
 * build the neural net.
 * @return a trained neural network.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
public Neural generateNeuralNet( List discriminators )
{

   // Get the number of output nodes
   numOutputNodes = discriminators.size();

   // Calculate the number of hidden layer nodes
   numHiddenNodes = numOutputNodes + this.NUM_INPUT_NODES / 2;

   // Create the binary output vectors
   List trainingOutputVectors = this.generateTrainingOutputVectors(numOutputNodes);

   // Create an instance of a neural network
   Neural localNeural = new Neural( this.NUM_INPUT_NODES,
   numHiddenNodes, numOutputNodes );

   // Convert the input vectors in a single large float array
   float[] inputs = this.convertVectorsIntoFloatArray( discriminators );

   // Convert the output vectors into a single large float array
   float[] outputs = this.convertVectorsIntoFloatArray( trainingOutputVectors );

   // Set the number of training cases
   int numTrainingCases = discriminators.size();
   localNeural.NumTraining = discriminators.size();

   // Create a local variable to track the output error of training
   float outputError = 0;

   // Create a local variable to track the number of training epochs gone past
   int epochCount = 0;

   // Perform the training until the max epocsh preference is reached or until
   // the output error is less than the output error tolerance.  Note: the training
   // process sets the weight matrices of the Neural Net object.
   for ( int i = 0; i < this.neuralMaxEpochs; i++ )
   {
      outputError = localNeural.Train( inputs, outputs, numTrainingCases,
      this.neuralLearningRate );
      epochCount++;

      if ( outputError < this.neuralOutputErrorTolerance )
      {
         break;
      }
   }

   System.out.println( "\n\nNetwork trained after " + epochCount + " epochs." );
   System.out.println( "\nError is: " + Float.toString( outputError ) );

   // return the trained neural netword object
   return localNeural;

}   // End generateNeuralNet
```

```
/** ****************************************************************************
 * <br>
 * Makes a formatted file to represent the trained neural network.
 *  Returns true if the
 * file io was successful, false otherwise.
 *
 * @param nn - a trained neural network.
 * @param path - the path to be used to save the neural network file to disk.
 * @return true if the file was created and saved, false otherwise.
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public boolean makeFile( Neural nn, String path )
{

   try
   {
      nn.Save( path );
      return true;
   }
   catch ( Exception e )
   {
      e.printStackTrace();
      return false;
   }

}   // End makeFile


/** ****************************************************************************
 * <br>
 * Returns a list of binary output vectors to be used in the training of
 * the neural network.  The number of output vectors is equal to the number of
 * discriminator vectors passed into the generateNeuralNet method. In this
 * implementation, each output vector is represented as an array of floats. The
 * ouput vectors contian a binary patterns that distinguish each output node. An
 * example for four output nodes is shown below:
 * <br><br>
 * output node 1 { 1.0, 0.0, 0.0, 0.0 }
 * output node 2 { 0.0, 1.0, 0.0, 0.0 }
 * output node 3 { 0.0, 0.0, 1.0, 0.0 }
 * output node 4 { 0.0, 0.0, 0.0, 1.0 }
 * <br><br>
 * Since the output nodes of the neural network represent the attribuates of the
 * FCR, the unique output patterns ensure that the neural network will match an
 * FCR attribute given the correct input.
 *
 * @param numOutputNodes - the number of output nodes for the neural network
 * @return A list containing the output vectors as float arrays
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public List generateTrainingOutputVectors( int numOutputNodes )
{

   List results = new ArrayList();

   for ( int nodeID = 0; nodeID < numOutputNodes; nodeID++ )
   {
      // Create a new float array
      float[] vector = new float[numOutputNodes];

      // Populate the array
      for ( int index = 0; index < vector.length; index++ )
      {
         if ( index == nodeID )
         {
```

```
            vector[index] = 1.0f;
         }
         else
         {
            vector[index] = 0.0f;
         }
      }
      // Add the newly created output vector to the list
      results.add( vector );
   }

   return results;

}   // End generateTrainingOuputVectors


/** ***************************************************************************
 * <br>
 * Returns a single float array representation of the list of vectors
 * passed in as the parameter.  This is utility function is required in order to
 * convert the data into a format recognizable by the mwa.ai.neural.Neural class.
 * <b>The list of input vectors must be float arrays in this
 * method.</b>  A function for
 * doubles can be easily created by copying this method.
 *
 * @param vectors - a list of discriminator vectors or desired output vectors.
 * @return A single float array resulting from the merger of the input vectors.
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
public float[] convertVectorsIntoFloatArray( List vectors )
{

   // Get the size of the first array in the list. The other arrays in the
   // list should be the same size given the process to get here.  An input list
   // is either for discriminator vectors, in which case the size of each array
   // is 28, or an list of deired output vectors, in which case the size of each
   // array is equal to the number of output vectors of the neural network.
   int arrayLength = ( ( float[] ) vectors.get( 0 ) ).length;

   // Create a singel dimension float array to hold all the arrays passed in
   // vectors list. This is the number of arrays in the list * the size
   // of each array.
   float[] results = new float[ vectors.size() * arrayLength ];

   // Create a singleton counter to help add floats to the results array
   int counter = 0;

   // Get an iterator for the input list
   Iterator vectorIter = vectors.iterator();

   while ( vectorIter.hasNext() )
   {
      // Get the float array contained in the list
      float[] vector = ( float[] ) vectorIter.next();

      // Add each element of the float array to our results array
      for ( int i = 0; i < vector.length; i++ )
      {
         results[counter++] = vector[i];
      }

   }

   return results;

}   // End makeFile
```

```
}       // End of NeuralNetGeneratorImpl
```

C. PACKAGE: mil.navy.nps.cs.babel.Correlator.semanticSearchEngine

1. Interface SemanticSearchEngine

```
//*****************************************************************************
// Filename:.............SemanticSearchEngine.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.semanticSearchEngine;

import java.util.List;
import java.util.Hashtable;

import mil.navy.nps.cs.oomi.fiom.CCR;


/** ************************************************************************
 * <br>
 * The <i>SemanticSearchEngine</i> interface defines the services that must be
 * provided by the a semantic search engine implementation to the component model
 * correlator.  The class <i>ComponentModelCorrelator</i> uses this interface to
 * perform the semantic correlation phase of the component model correlator.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 ************************************************************************** */
public interface SemanticSearchEngine
{

   /** ************************************************************************
    * <br>
    * Returns a double dimension array with the name of the FEV and
    * associated score for
    * every FCR passed into the method via the fcrList parameter.  The
    * format of the return
    * array is specific.  Each element of the array consists of a subArray
    * with the following
    * objects:
    * <br>
    * <br>
    * index 0:  The name of the FEV as a String object.
    * <br>
    * index 1:  The score of the FCR-CCR semantic comparison as a Float object.
    * <br>
    * <br>
    * The other members of the component model correlator rely on this
    * format and it cannot
    * be changed.
    *
    * @param fcrList - the list of FCRs to be semantically compared against the CCR
    * @param currentCCR - the candidate CCR for which correlation is desired
    * @return a double dimension array containing the FEV names and
    * corresponding scores.
    *
    * @author LT Steve Shedd
    *
    ************************************************************************** */
   public Object[] [] runSemanticSearch( List fcrList, CCR currentCCR );
```

```
}       // End SemanticSearchEngine
```

2. Class SemanticSearchEngineImpl

```
//*****************************************************************************
// Filename:.............SemanticSearchEngineImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.semanticSearchEngine;

import java.util.List;
import java.util.Iterator;
import java.util.Hashtable;
import java.util.ArrayList;

import mil.navy.nps.cs.oomi.fiom.CCR;
import mil.navy.nps.cs.oomi.fiom.FCR;


/** ***************************************************************************
 * <br>
 * The <i>SemanticSearchEngineImpl</i> class performs the semantic
 * correlation phase of the
 * component model correlator.  The meat of the correlation is contained within the
 * <i>runSemanticSearch</i> method.  Should future work require a change to the
 * implementation of the semantic correlation process, it is hoped
 * that only this class
 * will have to be modified.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 **************************************************************************** */
public class SemanticSearchEngineImpl implements SemanticSearchEngine
{


   /** Member variable to hold the results of the semantic search. */
   Object[] [] semanticResults = null;


   /** Default constructor */
   public SemanticSearchEngineImpl()
   {

   }


   /** ************************************************************************
    * <br>
    * Returns a double dimension array with the name of the FEV and
    * associated score for
    * every FCR passed into the method via the fcrList parameter.  The
    * format of the return
    * array is specific.  Each element of the array consists of a subArray
    * with the following
    * objects:
    * <br>
    * <br>
    * index 0:  The name of the FEV as a String object.
    * <br>
```

```
 * index 1:  The score of the FCR-CCR semantic comparison as a Float object.
 * <br>
 * <br>
 * The other members of the component model correlator rely on this
 * format and it cannot
 * be changed.
 *
 * @param fcrList - the list of FCRs to be semantically compared against the CCR
 * @param currentCCR - the candidate CCR for which correlation is desired
 * @return a double dimension array containing the FEV names and
 * corresponding scores.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public Object[] [] runSemanticSearch( List fcrList, CCR currentCCR )
{

   // Reset score variables
   float fcrScore = 0;

   // Size the results double dimension array
   semanticResults = new Object[ fcrList.size() ] [2];

   // Variable used in the while loop below to identify which row of the
   // semanticResults array to record the semantic score.
   int fcr = 0;

   // Get the CCR keywords
   String[] ccrKeywords = currentCCR.getCCRSemantics().getKeywordList();

   // Get an iterator for the FCR list
   Iterator fcrIter = fcrList.iterator();

   // For each FCR in the FIOM
   while ( fcrIter.hasNext() )
   {

      // Reset score variables
      fcrScore = 0;

      // Get a local copy of the FCR
      FCR localFCR = ( FCR )fcrIter.next();

      // Get the keywords for the FCR
      String[] fcrKeywords = localFCR.getFCRSemantics().getKeywordList();

      // Calculate the score for the FCR.  Total match count
      // divided by the total number of keywords for the CCR.
      fcrScore = runComparisonForScore( fcrKeywords, ccrKeywords );

      // Add the results to the semanticResults double dimention array.
      // The following schema is used for the array:
      //    index 0: FEV Name
      //    index 1: FCR Score
      semanticResults[fcr] [0] = new String( localFCR.getFCRName() );
      semanticResults[fcr] [1] = new Float( fcrScore );

      // Incremenet the FCR counter variable
      fcr++;

   }

   // Sort the results
   mergeSort( semanticResults, 0, semanticResults.length - 1 );

   // Return the semantic results array.
   return semanticResults;
```

```
}   // End runSemanticSearch


/** ****************************************************************************
 * <br>
 * Returns the semantic comparison score for the FCR and CCR comparison.
 * The implementation
 * of the comparison is contained within this method and can be changed
 * without affecting
 * the rest of the semantic search engine.
 *
 * @param fcrWords - the list of keywords for the FCR being tested
 * @param ccrWords - the list of keywords for the CCR being tested
 * @return a semantic comparison score for the FCR
 *
 * @author LT Steve Shedd
 *
 ***************************************************************************** */
private float runComparisonForScore( String[] fcrWords, String[] ccrWords )
{

   float fcrScore = 0;
   int matchCount = 0;

   // For each word in the FCR keyword list, compare it with each word
   // from the CCR keyword list.  Keep track of the matches and compute
   // a percentage score.
   for ( int i = 0; i < ccrWords.length; i++ )
   {
      for ( int j = 0; j < fcrWords.length; j++ )
      {
         if ( ( ( String )ccrWords[i] ).equalsIgnoreCase( fcrWords[j] ) )
         {
            matchCount += 1;
         }
      }
   }

   // Must cast the numerator and denominator to floats before the
   // division operation. Otherwise, the integer division will return the
   // floor of a number less than 1, which will always be 0.
   fcrScore = ( float )matchCount / ( float )ccrWords.length;

   return fcrScore;

}   // End runComparisonForScore


/** ****************************************************************************
 * <br>
 * Implementation of of the merge-sort algorithm for the double
 * dimension array used
 * to store the FEV/score pairs.  The first column in the array must
 * contain the names
 * of the FEVs and the second column must contain the scores as
 * floats. This is an
 * optimal sorting algorithm which executes in O(n lg n ) in the worst
 * case scenario. This
 * implementation sorts the array in descending order based on the
 * scores of the FEVs
 * contained in the array.
 *
 * @param A - a double dimension array containing the FEV/Score
 * results to be sorted
 * @param indexLo - The start index of the array
 * @param indexHi - The end index of the array
 *
 * @author LT Steve Shedd
 *
```

```
    ************************************************************************** */
   private void mergeSort( Object[] [] A, int indexLo, int indexHi )
   {

      int lo = indexLo;
      int hi = indexHi;

      if ( lo >= hi )
      {
         return;
      }

      // Determine the value of mid.
      // int mid = ( lo + hi ) / 2;

      // Determine the value of mid in double form.
      Double doubMid = new Double( Math.floor( ( lo + hi ) / 2 ) );

      // Get mid as an integer value
      int mid = doubMid.intValue();

      //Partition the list into two lists and sort them recursively
      mergeSort( A, lo, mid );
      mergeSort( A, mid + 1, hi );

      //Merge the two sorted lists
      int end_lo = mid;
      int start_hi = mid + 1;
      while ( ( lo <= end_lo ) && ( start_hi <= hi ) )
      {

         // Check the float values in the array.
         if ( ( ( Float )A[lo] [1] ).floatValue() >
             ( ( Float )A[start_hi] [1] ).floatValue() )
             {
                lo++;
         }
         else
         {
            /*
             *  a[lo] >= a[start_hi]
             *  The next element comes from the second list,
             *  move the a[start_hi] element into the next
             *  position and shuffle all the other elements up.
             */

            Object[] temp = A[start_hi];

            for ( int k = start_hi - 1; k >= lo; k-- )
            {
               A[k + 1] = A[k];
            }

            A[lo] = temp;
            lo++;
            end_lo++;
            start_hi++;
         }

      }

   }   // End mergeSort


}      // End SemanticSearchEngineImpl
```

D. PACKAGE: mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine

1. Interface SyntacticSearchEngine

```
//*****************************************************************************
// Filename:.............SyntacticSearchEngine.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine;

import java.util.List;

import mil.navy.nps.cs.oomi.fiom.CCR;


/** ************************************************************************
 * <br>
 * The <i>SyntacticSearchEngine</i> interface defines the services that must be
 * provided by the a syntactic search engine implementation to the component model
 * correlator.  The class <i>ComponentModelCorrelator</i> uses this interface to
 * perform the syntactic correlation phase of the component model correlator.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 *
 ************************************************************************** */
public interface SyntacticSearchEngine
{

   /** ************************************************************************
    * <br>
    * Returns a double dimension array with the name of the FEV and
    * associated score for
    * every FCR passed into the method via the fcrList parameter.  The
    * format of the return
    * array is specific.  Each element of the array consists of a subArray
    * with the following
    * objects:
    * <br>
    * <br>
    * index 0:  The name of the FEV as a String object.
    * <br>
    * index 1:  The score of the FCR-CCR syntactic comparison as a Float object.
    * <br>
    * <br>
    * The other members of the component model correlator rely on this
    * format and it cannot
    * be changed.
    *
    * @param fcrList - the list of FCRs to be syntactically compared against the CCR
    * @param currentCCR - the candidate CCR for which correlation is desired
    * @return a double dimension array containing the FEV names and
    * corresponding scores.
    *
    * @author LT Steve Shedd
    *
    ************************************************************************** */
   public Object[] [] runSyntacticSearch( List fcrList, CCR currentCCR );


}      // End SyntacticSearchEngine
```

2. Class SyntacticSearchEngineImpl

```
//*********************************************************************************
// Filename:.............SyntacticSearchEngineImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*********************************************************************************

package mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine;

import java.util.List;
import java.util.Iterator;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.Arrays;

import mwa.ai.neural.*;

import mil.navy.nps.cs.oomi.fiom.*;


/** ********************************************************************************
 * <br>
 * The <i>SynatacticSearchEngineImpl</i> class performs the syntactic
 * correlation phase of the
 * component model correlator.  The meat of the correlation is contained within the
 * <i>runSyntacticSearch</i> method.  Should future work require a change to the
 * implementation of the syntactic correlation process, it is hoped
 * that only this class
 * will have to be modified.
 *
 * @author LT Steve Shedd
 * @version v1.0 September 2002
 **********************************************************************************
@testcase
test.mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine.TestSyntacticSearchEngineImpl
 *
 ********************************************************************************** */
public class SyntacticSearchEngineImpl implements SyntacticSearchEngine
{

   /** Member variable to hold the results of the syntactic search. */
   Object[] [] syntacticResults = null;


   /** Default constructor */
   public SyntacticSearchEngineImpl()
   {

   }


   /** ********************************************************************************
    * <br>
    * Returns a double dimension array with the name of the FEV and
    * associated score for
    * every FCR passed into the method via the fcrList parameter.  The
    * format of the return
    * array is specific.  Each element of the array consists of a subArray
    * with the following
    * objects:
    * <br>
    * <br>
    * index 0:  The name of the FEV as a String object.
    * <br>
```

```
 * index 1:  The score of the FCR-CCR syntactic comparison as a Float object.
 * <br>
 * <br>
 * The other members of the component model correlator rely on this
 * format and it cannot
 * be changed.
 *
 * @param fcrList - the list of FCRs to be syntactically compared against the CCR
 * @param currentCCR - the candidate CCR for which correlation is desired
 * @return a double dimension array containing the FEV names and
 * corresponding scores.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
public Object[] [] runSyntacticSearch( List fcrList, CCR currentCCR )
{

   // Create a temporary data structure to hold search results
   List tempResults = new ArrayList();

   // Variables used for the CCR-FCR comparison matrix
   float rowMax                     = 0;
   float matrix2normScore           = 0;
   float fcrPercentageScore         = 0;
   float fcrAttribCount             = 0;
   float rowAttribMatchingThreshold = 0.8f;
   boolean isOneToOne               = true;

   // Get the list of CCR discriminator vectors
   List ccrVectors = currentCCR.getCCRSyntax().getDiscriminatorVectors();

   // Get an iterator for the list of FEVs passed into the
   // Syntactic Correlator from the Semantic Correlator
   Iterator fcrIter = fcrList.iterator();

   // Iterate through the list of FEVs.  For each FEV FCR, perform a
   // a forward pass of each CCR discriminator vector through the FCR's
   // trained neural network.  Tally the score using the CCR-FCR
   // comparison matrix variables.
   while ( fcrIter.hasNext() )
   {

      // Reset the CCR-FCR Comparison Matrix variables
      matrix2normScore = 0;
      fcrPercentageScore = 0;
      fcrAttribCount = 0;

      // Get the next FCR to test
      FCR testFCR = ( FCR )fcrIter.next();

      // Get the trained neural network for the test FCR
      Neural fcrNeuralNet = testFCR.getFCRSyntax().getNeuralNetwork();

      // Create and initializ a boolean tracking array the size of the number
      // of outputs of the neural net.  Since the number of output nodes
      // is equal to the number of FCR attributes, we can identify
      // which attribute corresponds to the row max of one entry in the
      // comparison matrix.  When the attribute is identified, we can
      // set the corresponding attribTracker value to true.  If the
      // attrib tracker contains any false entries at the end, we know
      // a one-to-one correspondence has not been achieved.
      boolean[] hasAttribMatch = new boolean[ fcrNeuralNet.NumOutputs ];

      for ( int i = 0; i < hasAttribMatch.length; i++ )
      {
         hasAttribMatch[i] = false;
      }
```

```
// Get an iterator for the CCR discriminator vector list
Iterator ccrVectorIter = ccrVectors.iterator();

while ( ccrVectorIter.hasNext() )
{

   // Reset the row maximum variable
   rowMax = 0;

   // Create a local variable to track which attribute
   // has the row max.
   int attrib = -1;

   // Set the input vector for the forward pass
   fcrNeuralNet.Inputs = ( float[] ) ccrVectorIter.next();

   // Perform the forward pass.  This method uses the input vector
   // set in the previous line.
   fcrNeuralNet.ForwardPass();

   // Cycle through the neural net output
   // vector and determine the row Maximum
   for ( int j = 0; j < fcrNeuralNet.OutputLayer_Out.length; j++ )
   {
      float val = fcrNeuralNet.OutputLayer_Out[j];

      if ( rowMax < val )
      {
         rowMax = val;

         if ( rowMax > rowAttribMatchingThreshold )
         {
            attrib = j;
         }
      }
   }

   // Set the attribTracker element that matches the FCR
   // attribute to true.
   if ( attrib != -1 )
   {
      hasAttribMatch[attrib] = true;
   }

   // Add the square of the row max to the running tally of
   // the matrix score.
   matrix2normScore += ( float )Math.pow( ( double )rowMax, 2 );

}

// Complete the calculation of the 2-Norm for the
// CCR-FCR comparison matrix
matrix2normScore = ( float )Math.sqrt( ( double )matrix2normScore );

// Compute a percentage score for the FCR being tested.
fcrPercentageScore = ( float )( matrix2normScore * 100 / 2.0 );

// Determine if the CCR-FCR comparison is one-to-one
for ( int k = 0; k < hasAttribMatch.length; k++ )
{
   if ( !hasAttribMatch[k] == false )
   {
      isOneToOne = false;
      break;
   }
}

// If the correspondence is one-to-one, add the results to the
// syntacticResults double dimention array.
```

```
      // The following schema is used for the array:
      //    index 0: FEV Name
      //    index 1: FCR Score
      if ( isOneToOne )
      {
         // Create a single FEV/Score pair
         Object[] singleResult = new Object[2];
         singleResult[0] = new String( testFCR.getFEV().getFEVName() );
         singleResult[1] = new Float( fcrPercentageScore );

         // Add the pair to the temporary results list
         tempResults.add( singleResult );
      }

   }


   // If there are results, convert the temporary list into
   // a double dimension object array.
   if ( tempResults.size() > 0 )
   {
      // Size the results double dimension array
      syntacticResults = new Object[ tempResults.size() ] [2];

      // Create an counter to place items in the final results
      int fev = 0;

      Iterator tempResultsIter = tempResults.iterator();

      while ( tempResultsIter.hasNext() )
      {

         // Copy the single result pair into the syntacticResults Array
         Object[] fevScorePair = ( Object [] ) tempResultsIter.next();

         syntacticResults[fev] [0] = fevScorePair[0];
         syntacticResults[fev] [1] = fevScorePair[1];

      }

      // Order the results according to score
      mergeSort( syntacticResults, 0, syntacticResults.length - 1 );

   }
   else
   {
      // If there are no results, then create a single entery in the array
      // and return it.
      syntacticResults = null;
   }

   //return syntacticResults;
   return syntacticResults;

}   // End runSyntacticSearch


/** ****************************************************************************
 * <br>
 * Implementation of of the merge-sort algorithm for the double
 * dimension array used
 * to store the FEV/score pairs.  The first column in the array must
 * contain the names
 * of the FEVs and the second column must contain the scores as
 * floats. This is an
 * optimal sorting algorithm which executes in O(n lg n ) in the
 * worst case scenario.
 *
 * @param A - a double dimension array containing the FEV/Score
```

```
 * results to be sorted
 * @param indexLo - The start index of the array
 * @param indexHi - The end index of the array
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
private void mergeSort( Object[] [] A, int indexLo, int indexHi )
{

   int lo = indexLo;
   int hi = indexHi;

   if ( lo >= hi )
   {
      return;
   }

   // Determine the value of mid.
   // int mid = ( lo + hi ) / 2;

   // Determine the value of mid in double form.
   Double dmid = new Double( Math.floor( ( lo + hi ) / 2 ) );

   // Get mid as an integer value
   int mid = dmid.intValue();

   //Partition the list into two lists and sort them recursively
   mergeSort( A, lo, mid );
   mergeSort( A, mid + 1, hi );

   //Merge the two sorted lists
   int end_lo = mid;
   int start_hi = mid + 1;
   while ( ( lo <= end_lo ) && ( start_hi <= hi ) )
   {

      if ( ( ( Float )A[lo] [1] ).floatValue() <
      ( ( Float )A[start_hi] [1] ).floatValue() )
      {
         lo++;
      }
      else
      {
         /*
          *  a[lo] >= a[start_hi]
          *  The next element comes from the second list,
          *  move the a[start_hi] element into the next
          *  position and shuffle all the other elements up.
          */

         Object[] temp = A[start_hi];

         for ( int k = start_hi - 1; k >= lo; k-- )
         {
            A[k + 1] = A[k];
         }

         A[lo] = temp;
         lo++;
         end_lo++;
         start_hi++;
      }

   }

}   // End mergeSort
```

```
}       // End SyntacticSearchEngine
```

E. PACKAGE: mil.navy.nps.cs.babel.Correlator

1. Class CMCorrelatorAdaptor

```
//*****************************************************************************
// Filename:.............CMCorrelatorAdaptor.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN and LT George Lawler, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator;

import mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface;
import javax.swing.JComponent;

import org.jdom.Document;


public class CMCorrelatorAdaptor implements CMCorrelatorInterface
{

   public CMCorrelatorAdaptor()
   {
   }


   public void updateCCRList()
   {
      /** ***************************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
       * method
       ***************************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method updateCCRList() not yet
implemented." );
   }


   public JComponent getCorrelatorPanel()
   {
      /** ***************************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ***************************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method getCorrelatorPanel() not
yet implemented." );
   }


   public void generateCCRSemanticComponents( Document paramJDOMDoc )
   {
      /** ***************************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ***************************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method
generateCCRSemanticComponents() not yet implemented." );
   }


   public void generateCCRSyntacticComponents( Document paramJDOMDoc )
   {
      /** ***************************************************************************
```

```
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ********************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method
generateCCRSyntacticComponents() not yet implemented." );
   }


   public void generateFCRSemanticComponents( Document paramJDOMDoc )
   {
      /** ********************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ********************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method
generateFCRSemanticComponents() not yet implemented." );
   }


   public void generateFCRSyntacticComponents( Document paramJDOMDoc )
   {
      /** ********************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ********************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method
generateFCRSyntacticComponents() not yet implemented." );
   }


   public void doSyntacticCorrelation()
   {
      /** ********************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ********************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method doSyntacticCorrelation()
not yet implemented." );
   }


   public void doSemanticCorrelation()
   {
      /** ********************************************************************
       *@todo: Implement this mil.navy.nps.cs.babel.connectors.CMCorrelatorInterface
method
       ********************************************************************** */
      throw new java.lang.UnsupportedOperationException( "Method doSemanticCorrelation()
not yet implemented." );
   }

}      //end CMCorrelatorAdapter
```

2. Class ComponentModelCorrelator

```
//*****************************************************************************
// Filename:.............ComponentModelCorrelator.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN and LT George Lawler, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
// Notes:................This class was originally created by LT George Lawler
// who is currently the master developer for the overall OOMI IDE user interface.
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator;
```

```
import javax.swing.JComponent;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.Hashtable;

import mil.navy.nps.cs.babel.Correlator.CMCorrelatorAdaptor;

import org.jdom.Document;

import mil.navy.nps.cs.babel.event.CorrelatorPanelEvent;
import mil.navy.nps.cs.babel.event.CorrelatorPanelEventListener;
import javax.swing.event.EventListenerList;
import mil.navy.nps.cs.babel.event.CurrentSelectionChangeListener;
import mil.navy.nps.cs.babel.event.CurrentSelectionChangeEvent;
import mil.navy.nps.cs.babel.connectors.OOMIDisplayInterface;

import mil.navy.nps.cs.babel.constructionManager.FIOMConstructionManager;
import mil.navy.nps.cs.babel.connectors.FIOMDBDirectInterface;
import mil.navy.nps.cs.babel.data.FIOMDatabase;

import mil.navy.nps.cs.babel.Correlator.semanticComponentGenerator.*;
import mil.navy.nps.cs.babel.Correlator.semanticSearchEngine.*;
import mil.navy.nps.cs.babel.Correlator.syntacticComponentGenerator.*;
import mil.navy.nps.cs.babel.Correlator.syntacticSearchEngine.*;

import mwa.ai.neural.*;


/** ****************************************************************************
 * <br>
 * The <i>ComponentModelCorrelator</i> is the overall executive of the Component
 * Model Correlator module of the OOMI IDE.  This class coordinates the following:
 * <br><br>
 * <ul>
 * <li> Generation of semantic correlation components
 * <li> Generation of syntactic correlation components
 * <li> Semantic correlation
 * <li> Syntactic correlation
 * </ul>
 * <br>
 *
 *
 * @author LT Steve Shedd, LT George Lawler
 * @version 1.0
 *
 ******************************************************************************* */

public class ComponentModelCorrelator extends CMCorrelatorAdaptor implements
CorrelatorPanelEventListener, CurrentSelectionChangeListener
{

   private CorrelatorPanel cmCorrelatorPanel = null;

   protected static FIOMDBDirectInterface handleForFIOMDB = null;

   private EventListenerList listenerList = new EventListenerList();

   private OOMIDisplayInterface handleForGUI = null;


   // Results data structures
   private ArrayList semanticResultsFCRList = new ArrayList();

   private Object[] [] filteredSemanticResults = null;
```

```
private Object[] [] filteredSyntacticResults = null;

private Object[] [] filteredCombinedResults = null;

// Correlator Properties and Preferences
private int semanticThreshold = 70;

private int syntacticThreshold = 70;

private int combinedThreshold = 70;

private float nueralLearningRate = 0.25f;

private int neuralMaxEpochs = 100000;

private double neuralOutputErrorTolerance = 0.1;



public ComponentModelCorrelator( OOMIDisplayInterface paramGUI )
{

   // Reference to the OOMI IDE display interface which can be used to send
   // status messages to the GUI.
   handleForGUI = paramGUI;

   //Singleton on the data base, only one FIOMDatabase
   handleForFIOMDB = FIOMDatabase.getDatabase();
   handleForFIOMDB.addCurrentSelectionChangeListener( this );

   cmCorrelatorPanel = new CorrelatorPanel( this );

}   //end constructor


//This class will handle all events from the displayed Correlator.  So
//  button actions get redirected here.

 /*  These legacy methods represent the functionality that must be captured
     by the event listener model in use by the correlator panel.  The
     methods used to use parameters, so the event object will have to
     carry as payload the values that used to be parameters.

     public void correlatorPreferencesButtonPressed();

     public void correlatorKeywordsButtonPressed(Object paramThreshold);

     public void correlatorNeuralNetButtonPressed(Object paramThreshold);

     public void correlatorCombinedButtonPressed(Object paramThreshold);

     public void correlatorShowSelectedFE();
 */


/** ****************************************************************************
 *
 *
 * @author  LT George Lawler, USN
 *
 ******************************************************************************* */
public void panelEventOccured( CorrelatorPanelEvent event )
{
   //System.out.println("something happened atleast");
   //System.out.println(event.getEventID());

   switch ( event.getEventID())
   {
```

```
         case CorrelatorPanelEvent.PREFERENCES_BUTTON_PRESSED :
            //System.out.println("Set preferences");
            break;
         case CorrelatorPanelEvent.KEYWORD_BUTTON_PRESSED :
            System.out.println("Redirecting KEYWORD_BUTTON event to
doSemanticCorrelation()");
            doSemanticCorrelation();
            break;
         case CorrelatorPanelEvent.NEURAL_NET_BUTTON_PRESSED :
            doSyntacticCorrelation();
            break;
         case CorrelatorPanelEvent.COMBINED_BUTTON_PRESSED :
            // Not sure if we still want to keep this in here
            // the comnbined button maybe should go away as well.
            break;
         case CorrelatorPanelEvent.SHOW_FEV_BUTTON_PRESSED :
            // Show selected FEV to the Register Tab Panel.
            break;
         case CorrelatorPanelEvent.CCR_SELECTED :
            this.handleForFIOMDB.setCurrentCCR(
            (String)((CorrelatorPanel)event.getSource()).ccrSelected.getSelectedItem());
            break;
         default :
            System.out.println( "Default envoked" );
            break;
      }


   }   //end panelEventOccured


   /** ****************************************************************************
    * This method forces the correlator panel to get the latest
    * Unregisterd CCR list.
    ***************************************************************************** */
   public void updateCCRList()
   {
      this.cmCorrelatorPanel.setCCRList( this.handleForFIOMDB.getCCRList());
   }


   /** Just returns the correlator panel that is held in this class */
   public JComponent getCorrelatorPanel()
   {
      return ( ( JComponent )this.cmCorrelatorPanel );
   }   //end getCorrelatorPanel


   /** ****************************************************************************
    * <br>
    * Creates the semantic components for a CCR needed for the semantic correlation.
    *
    * @param ccrJdomDoc - a JDOM Document representation of the CCRs
    * XML Schema file.
    * @return None
    *
    * @author LT Steve Shedd
    * @version 1.0, September 2002
    *
    ***************************************************************************** */
   public void generateCCRSemanticComponents( Document ccrJdomDoc )
   {

      // Create a temporary data structure
      String[] keywords = null;

      // Get an instance of a keyword generator
      KeywordGenerator semanticGen = new KeywordGeneratorImpl();
```

```
      try
      {
         // Get the keywords for the CCR
         keywords = semanticGen.generateKeywords( ccrJdomDoc );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }

      // Associate the keywords to the CCRSemantics component
      this.handleForFIOMDB.getCurrentCCR().getCCRSemantics().setKeywordList( keywords );

      System.out.print("\n" + handleForFIOMDB.getCurrentCCR().getCCRName() + " Keywords:
");
      for ( int i = 0; i < keywords.length; i++ )
      {
         System.out.print( keywords[i] + " ");
      }
      System.out.println();

      // Send a status message back to the OOMI IDE
      this.handleForGUI.setStatusBarText( "CCR Semantic Components Generated" );


   }   //end generateCCRSemanticComponents



   /** ****************************************************************************
    * <br>
    * Creates the syntactic components for a CCR needed for the
    * syntactic correlation.
    *
    * @param ccrJdomDoc - a JDOM Document representation of the CCRs
    * XML Schema file.
    * @return None
    *
    * @author LT Steve Shedd
    * @version 1.0, September 2002
    *
    **************************************************************************** */
   public void generateCCRSyntacticComponents( Document ccrJdomDoc )
   {

      // Create an instance of the Discriminator Generator
      DiscriminatorGenerator gen = new DiscriminatorGeneratorImpl();

      // Create a list for the discriminator vectors
      List discr = null;

      try
      {
         // run the generator
         discr = gen.generateDiscriminatorVectors( ccrJdomDoc );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }

      // Associate the discriminator vectors with the CCRSyntax component
      handleForFIOMDB.getCurrentCCR().getCCRSyntax().setDiscriminatorVectors( discr );

      Iterator vectors = discr.iterator();

      int i = 0;
```

```
      System.out.println("\n\n-----------" + handleForFIOMDB.getCurrentCCR().getCCRName()
+ "-----------");

      while ( vectors.hasNext() ) {

         float[] local = (float[])vectors.next();

         System.out.print("\nnorm" + (++i) + ": ");

         for ( int j = 0; j < local.length ; j++ ) {

            System.out.print( local[j] + " ");

         }
      }


      // Send a status message back to the GUI
      this.handleForGUI.setStatusBarText( "CCR Syntactic Components Generated" );


   }   //end generateCCRSyntacticComponents


   /** ****************************************************************************
    * <br>
    * Creates the semantic components for an FCR needed for the
    * semantic correlation.
    *
    * @param fcrJdomDoc - a JDOM Document representation of the FCRs
    * XML Schema file.
    * @return None
    *
    * @author LT Steve Shedd
    * @version 1.0, September 2002
    *
    **************************************************************************** */
   public void generateFCRSemanticComponents( Document fcrJdomDoc )
   {

      // Create a temporary data structure
      String[] keywords = null;

      // Get an instance of a keyword generator
      KeywordGenerator semanticGen = new KeywordGeneratorImpl();

      try
      {
         // Get the keywords for the FCR
         keywords = semanticGen.generateKeywords( fcrJdomDoc );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }

      // Associate the keywords to the FCRSemantics component
      this.handleForFIOMDB.getCurrentFCR().getFCRSemantics().setKeywordList( keywords );

      System.out.print("\n" + handleForFIOMDB.getCurrentFCR().getFCRName() + " Keywords:
");
      for ( int i = 0; i < keywords.length; i++ )
      {
         System.out.print( keywords[i] + " ");
      }
      System.out.println();

      // Send a status message back to the OOMI IDE
      this.handleForGUI.setStatusBarText( "FCR Semantic Components Generated" );
```

```
   }   //end generateFCRSemanticComponents


   /** ****************************************************************************
    * <br>
    * Creates the syntactic components for an FCR needed for the
    * syntactic correlation.
    *
    * @param fcrJdomDoc - a JDOM Document representation of the FCRs
    * XML Schema file.
    * @return None
    *
    * @author LT Steve Shedd
    * @version 1.0, September 2002
    *
    **************************************************************************** */
   public void generateFCRSyntacticComponents( Document fcrJdomDoc )
   {

      // Create an instance of the Discriminator Generator
      DiscriminatorGenerator gen = new DiscriminatorGeneratorImpl();

      // Create a list for the discriminator vectors
      List discr = null;

      try
      {
         // run the discriminator generator
         discr = gen.generateDiscriminatorVectors( fcrJdomDoc );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }

      // Associate the discriminator vectors with the CCRSyntax component
      handleForFIOMDB.getCurrentFCR().getFCRSyntax().setDiscriminatorVectors( discr );

      Iterator vectors = discr.iterator();

      int i = 0;

      System.out.println("\n\n-----------" + handleForFIOMDB.getCurrentFCR().getFCRName()
+ "-----------");

      while ( vectors.hasNext() ) {

         float[] local = (float[])vectors.next();

         System.out.print("\nnorm" + (++i) + ": ");

         for ( int j = 0; j < local.length ; j++ ) {

            System.out.print( local[j] + " ");

         }
      }

      // Create an instance of the neural net generator
      NeuralNetGenerator nnGen = new NeuralNetGeneratorImpl();

      // Create and train a neural network for the FCR
      Neural nn = nnGen.generateNeuralNet( discr );

      // Set the path for the neural network file
      String path = "c:\\root\\etc\\babel\\generated\\neural\\";
```

```
      path = path + handleForFIOMDB.getCurrentFCR().getFCRName();

      // Create and save a formated neural network file
      nnGen.makeFile( nn, path );

      // Associate the syntactic components with the FCRSyntax component
      handleForFIOMDB.getCurrentFCR().getFCRSyntax().setDiscriminatorVectors( discr );
      handleForFIOMDB.getCurrentFCR().getFCRSyntax().setNeuralNetwork( nn );
      handleForFIOMDB.getCurrentFCR().getFCRSyntax().setNNfilePath( path );

      // Send a status message to the OOMI IDE
      this.handleForGUI.setStatusBarText( "FCR Syntactic Components Generated" );

   }   //end generateFCRSyntacticComponents



   /** ****************************************************************************
    * <br>
    * Implementation of the semantic correlation phase of the Component Model
    * Correlator. This
    * method performs the semantic correlation between a CCR and the
    * list of FEV FCRs in
    * the FIOM.  It then sends the results of the correlation to the
    * results component of
    * the Correlator Panel in the OOMI IDE.
    *
    * @param None
    * @return None
    *
    * @author LT Steve Shedd
    * @version 1.0, September 2002
    *
    ***************************************************************************** */
   public void doSemanticCorrelation()
   {

      // Because the syntactic correlator depends on the results of the semantic
      // correlator, we null any previous syntactic results.
      this.filteredSyntacticResults = null;
      this.filteredSemanticResults = null;
      this.filteredCombinedResults = null;

      // Create an instance of the semantic search engine
      SemanticSearchEngine semanticSearch = new SemanticSearchEngineImpl();

      // Create a reference for the unfiltered search results
      Object[] [] unfilteredResults = null;

      // Get a list of all FCRs in the FIOM
      List localFIOMFCRList = this.handleForFIOMDB.getFCRList();

      // Check to see if there are FCRs in the FIOM
      if ( localFIOMFCRList != null )
      {
         // Perform the semantic correlation.  This will produce a sorted object
         // array of FCR names and Scores for every FCR in the FIOM
         unfilteredResults = semanticSearch.runSemanticSearch( localFIOMFCRList,
         handleForFIOMDB.getCurrentCCR());
      }
      else
      {
         this.handleForGUI.setStatusBarText( "No FEVs in FIOM" );
      }

      System.out.println("\n\nCurrent CCR is: " +
handleForFIOMDB.getCurrentCCR().getCCRName());
      System.out.println("Raw semantic results:\n");
      for ( int i = 0; i < unfilteredResults.length; i++ )
```

```
      {
         System.out.println( unfilteredResults[i] [0] + ": " + unfilteredResults[i] [1]
);
      }

      // Filter by Threshold value.  This will send any previous semantic
      // results to Java's Garbage collector.
      this.filteredSemanticResults =
      filterResultsByThreshold( unfilteredResults, this.semanticThreshold );

      // Save the list of resulting FEV names to the semantic FEV Results list.
      // This step is necessary for two reasons.  First, the results of the semantic
      // search are passed into the syntactic search.  However, the double dimension
      // array semanticResults contains an FEV/Score pair for every FEV in the FIOM.
      // Therefore, it is necessary to create a new list of the just hte FEV names.
      // Second, the semanticResults array can be re-filtered dynamically by the
      // interoperability engineer within the correlator panel. Therefore, a list
      // of filtered results must be kept separate from the aggregate results.
      for ( int i = 0; i < this.filteredSemanticResults.length; i++ )
      {
         this.semanticResultsFCRList.add( this.filteredSemanticResults[ i ] [ 0 ] );
      }

      System.out.println("\nFiltered semantic results:\n");
      for ( int i = 0; i < filteredSemanticResults.length; i++ )
      {
         System.out.println( filteredSemanticResults[i] [0] + ": " +
filteredSemanticResults[i] [1] );
      }
      System.out.println();


      // The combined results simply equals the semantic results at this point because
      // the syntactic correlation has not taken place.
      this.filteredCombinedResults = this.filteredSemanticResults;

      // Send the results to the correlator panel display
      this.sendResultsToCorrelatorPanel( this.filteredSemanticResults,
      this.filteredSyntacticResults,
      this.filteredCombinedResults );

      // Send a status message back to the OOMI IDE
      this.handleForGUI.setStatusBarText( "Semantic Correlation Completed" );


   }   // End doSemanticCorrelation


   /** ****************************************************************************
    * <br>
    * Implementation of the syntactic correlation phase of the Component Model
    * Correlator. This
    * method performs the syntactic correlation between a CCR and the
    * list of FEV FCRs in
    * the FIOM.  It then sends the results of the correlation to the
    * results component of
    * the Correlator Panel in the OOMI IDE.
    *
    * @param None
    * @return None
    *
    * @author LT Steve Shedd
    * @version 1.0, September 2002
    *
    **************************************************************************** */
   public void doSyntacticCorrelation()
   {

      // Create an instance of the syntactic search engine
```

```
      SyntacticSearchEngine syntacticSearch = new SyntacticSearchEngineImpl();

      // Create a reference for the unfiltered search results
      Object[] [] unfilteredResults = null;

      // Check to see if the semantic correlation returned some results
      if ( semanticResultsFCRList != null )
      {
         // Perform the semantic correlation.  This will produce a sorted object
         // array of FCR names and Scores for every FCR in list populated from
         // the semantic correlation results
         unfilteredResults = syntacticSearch.runSyntacticSearch( semanticResultsFCRList,
         handleForFIOMDB.getCurrentCCR());

      }
      else
      {

         // If there are no results from the semantic correlator, do the syntactic
         // correlation with the FCR list from the entire FIOM.

         List localFIOMFCRList = this.handleForFIOMDB.getFCRList();

         if ( localFIOMFCRList != null )
         {
            unfilteredResults = syntacticSearch.runSyntacticSearch( localFIOMFCRList,
            handleForFIOMDB.getCurrentCCR());
         }
         else
         {
            this.handleForGUI.setStatusBarText( "No FEVs in FIOM" );
         }

      }


      if ( unfilteredResults != null )
      {


          // Filter by Threshold value
          this.filteredSyntacticResults = filterResultsByThreshold( unfilteredResults,
          this.syntacticThreshold );

          // The combined results equals the average of the semantic and syntactic
scores.
          for ( int i = 0; i < this.filteredSemanticResults.length; i++ )
          {
             float sem = ( ( Float )this.filteredSemanticResults[ i ] [ 1 ]
).floatValue();
             float syn = ( ( Float )this.filteredSyntacticResults[ i ] [ 1 ]
).floatValue();
             float avg = ( sem + syn ) / 2;

             this.filteredCombinedResults[ i ] [ 1 ] = new Float( avg ) ;
          }

      }

      // Send the results to the correlator panel display
      this.sendResultsToCorrelatorPanel( this.filteredSemanticResults,
      this.filteredSyntacticResults,
      this.filteredCombinedResults );


   }   // End doSyntacticCorrelation


   /** ****************************************************************************
```

```
 * Utility function to filter a list of FEV/Score pairs by
 * threshold value setting.
 * This function returns a new double dimension array. The array must
 * have an FEV name
 * in index 0 and an FEV correlation score as index 1
 *
 * @param input - double dimension array containing FEV/Score pairs.
 * @param threshold - The threshold setting for the particular correlation
 * @return A new double dimension array containing the FEV/Score pairs
 * above the threshold
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************** */
private Object[] [] filterResultsByThreshold( Object[] [] input, int threshold )
{

   // Create a new structure for the filtered output
   Object[] [] filteredOutput = null;

   // Create a temporary data structure for the FEV names
   ArrayList FEVNames = new ArrayList();

   // Create a temporary data structure for the FEV scores
   ArrayList Scores = new ArrayList();

   float floatThreshold = (float)threshold/100f;

   // Cycle through the input double dimension array and
   // copy pairs above the threshold.
   for ( int i = 0; i < input.length; i++ )
   {
      if ( ( ( Float )input[ i ] [ 1 ] ).floatValue() >= floatThreshold )
      {
         FEVNames.add( i, input[ i ] [ 0 ] );
         Scores.add( i, input[ i ] [ 1 ] );
      }
   }

   // Size the filtered output double dimension array
   filteredOutput = new Object[ FEVNames.size() ] [ 2 ];

   //Convert the FEVNames and Scores lists to a double dimension object array
   for ( int j = 0; j < FEVNames.size(); j++ )
   {
      // Copy the FEV Name
      filteredOutput[ j ] [ 0 ] = new String((String)FEVNames.get( j ));

      // Copy the associated FEV Score
      filteredOutput[ j ] [ 1 ] = new Float(((Float)Scores.get( j )).floatValue());
   }

   return filteredOutput;

}   // End filterResultsByThreshold


/** ********************************************************************************
 * <br>
 * Formats the FEV/Score pairs produced by the semantic and syntactic
 * correlators for the
 * Correlator Panel.  The Correlator Panel uses a JTable to display the
 * correlation results.
 * The defaultTableModel for a JTable can be used to convert a double
 * dimension array into
 * a JTable.  This method merges the three input double dimension arrays
 * into a single double
 * dimension array.  The JTable in the correlator panel is then populated
 * with the results.
```

```
 * The pairs in the input array must have an FEV name in index 0 and
 * the FEV score in
 * index 1.
 *
 * @param semanticResults - the filtered results of the semantic correlation
 * @param syntacticResults - the filtered results of the syntactic correlation.
 * @param combinedResults - the filtered combined correlation results.
 *
 * @author LT Steve Shedd
 *
 ********************************************************************************* */
private void sendResultsToCorrelatorPanel( Object[] [] semanticResults,
Object[] [] syntacticResults, Object[] [] combinedResults )
{

   // Create a new array for the table data.  Size the array with
   // the length of the semantic results since the semantic results
   // are passed on to the syntactic correlator.
   Object[] [] tableData = new Object[ semanticResults.length ] [ 6 ];


   for ( int i = 0; i < semanticResults.length; i++ )
   {

      tableData[ i ] [ 0 ] = semanticResults[ i ] [ 0 ];
      tableData[ i ] [ 1 ] = semanticResults[ i ] [ 1 ];
      tableData[ i ] [ 2 ] = "Pass-through";

      if ( syntacticResults == null )
      {
         tableData[ i ] [ 3 ] = null;
      }
      else
      {
         tableData[ i ] [ 3 ] = syntacticResults[ i ] [ 1 ];
      }

      tableData[ i ] [ 4 ] = combinedResults[ i ] [ 1 ];
      tableData[ i ] [ 5 ] = "FE Name";

   }

   // Create defealt Table model with the table data array.
   this.cmCorrelatorPanel.setContent( tableData );

   // Remains to be implemented

}


// The correlator panel works with the CCR list, and so it must
// know when the current CCR is changed, this listener method
// will be called when ever the CurrentCCR is changed in the
// FIOMDatabase class.
public void selectionChanged( CurrentSelectionChangeEvent event )
{
   if ( event.getSelectionThatChanged() ==
   CurrentSelectionChangeEvent.CCR_SELECTION_CHANGED )
   {
      //Update the CCR list in the correlator panel
      this.updateCCRList();
   }
   else
   {
      // no action for Current FCR change event
   }
}   //end selectionChanged
```

```
}       //end ComponentModelCorrelator
```

3. Class CorrelatorPanel

```
//*****************************************************************************
// Filename:.............ComponentModelCorrelator.java
// Parent Project:.......OOMI IDE
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT George Lawler, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
// Notes:................This class was originally created by LT George Lawler
// during the development of the OOMI IDE prior to the completion of the features
// of the Component Model Correlator.
//*****************************************************************************

package mil.navy.nps.cs.babel.Correlator;

import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.*;

import mil.navy.nps.cs.babel.event.CorrelatorPanelEvent;
import mil.navy.nps.cs.babel.event.CorrelatorPanelEventListener;
import javax.swing.event.EventListenerList;

import javax.swing.JTable;
import javax.swing.table.TableModel;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
//import javax.swing.table.TableColumn;


/** ************************************************************************
 * Title:       Babel Correlator Panel
 * Description: Front end for Register Tab Panel connection to the Correlator.
 *              Also provides area for display of correlator results.
 *
 *              Pass in the MainFrame parametere during construction so that
 *              this panel can access the controlChannel to pass events to the
 *              back end.
 *
 * @author      LT George Lawler
 ************************************************************************** */
public class CorrelatorPanel extends JPanel
{

   //This reference allows access to the controlChannel, which is of
   //  type OOMIControlInterface, in this case.
   //private MainFrame parentFrame = null;


   private JPanel contentPane = null;


   /** ************************************************************************
    * The table used to display the results of the component model
    * correlation phases.
    ************************************************************************** */
   JTable table = null;


   /** ************************************************************************
    * The title for the first column in the table displayed on this panel.
    * This title is built into a TableModel that is used to construct the
    * table.
    ************************************************************************** */
```

```
public static String COLUMN_ZERO_TITLE = new String( "FEV" );


/** ****************************************************************************
 * The title for the second column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 ***************************************************************************** */
public static String COLUMN_ONE_TITLE = new String( "Keyword Score" );


/** ****************************************************************************
 * The title for the third column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 ***************************************************************************** */
public static String COLUMN_TWO_TITLE = new String( "Pass-Thru" );


/** ****************************************************************************
 * The title for the fourth column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 ***************************************************************************** */
public static String COLUMN_THREE_TITLE = new String( "Neural Net Score" );


/** ****************************************************************************
 * The title for the fifth column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 ***************************************************************************** */
public static String COLUMN_FOUR_TITLE = new String( "Combined Score" );


/** ****************************************************************************
 * The title for the sixth column in the table displayed on this panel.
 * This title is built into a TableModel that is used to construct the
 * table.
 ***************************************************************************** */
public static String COLUMN_FIVE_TITLE = new String( "Select FE" );


// ** This next section is indented to show which parts are 'contained'
//     in which of the overal panel's sub containers.

//This control area panel holds all the widgets
JPanel controlArea = new JPanel();

JPanel ccrSelectionSubPanel = new JPanel();

JLabel ccrSelectionLabel = new JLabel();

JComboBox ccrSelected = new JComboBox();

JButton preferencesButton = new JButton();

JPanel thresholdSubPanel = new JPanel();

JLabel buttonLabel = new JLabel();

JButton  filterOnKeyword = new JButton();

JButton filterOnNeuralNet = new JButton();

JButton filterOnCombined = new JButton();

JLabel comboBoxLabel = new JLabel();
```

```
JComboBox keywordThreshold = null; //new JComboBox((();

JComboBox neuralNetThreshold = null;      //new JComboBox();

JComboBox combinedScoreThreshold = null;  //new JComboBox();

JPanel actionSubPanel = new JPanel();

JButton showSelectedFE = new JButton();

private Insets buttonInsets = new Insets( 2,   //top
    2,    //left
    2,   //bottom
    2 );   //right


//This data area panel holds the table that shows the resluts
JScrollPane dataArea = null;


//This was the first way I made the table. Its not very efficient
//TableColumn fevColumn = new TableColumn();
//TableColumn keywordScoreColumn = new TableColumn();
//TableColumn passThruOptionColumn = new TableColumn();
//TableColumn neuralNetScoreColumn = new TableColumn();
//TableColumn combinedScoreColumn = new TableColumn();
//TableColumn selectFEColumn = new TableColumn();



//This is how many choices to put in the Threshold drop down arrays
private static int NUMBER_OF_VALUES = 10;


//This is the distance between thresholds.  Increment is
//  multiplied by the index plus one (plus one so the first
//  value isn't zero).  NUMBER_OF_VALUES times the INCREMENT
//  is the maximum value in the table.
private static int INCREMENT = 10;

// so that the value shown is 70
private static int DEFAULT_THRESHOLD_INDEX = 6;

//this Object array holds strings that represent the
//  integer value for the search threshold
private Object[] thresholdValues = null;


private EventListenerList listenerList = new EventListenerList();


/** ****************************************************************************
 * Constructor calls the private init method to set up the
 * panel, and catches any exceptions encountered during panel
 * setup.
 * @param MainFrame reference used to gain access to the OOMIControlInterface
 *
 * Pass in to this object a CCR List and an FCR list so that the panel
 * will come up with a few things in it for the demo.
 *
 * Hold off on constructing this panel until the FIOM is created, so that
 * there will be values available to pass as parameters.
 ******************************************************************************* */
public CorrelatorPanel( CorrelatorPanelEventListener paramEventListener )
{

   //parentFrame = paramParent;

   try
```

```
   {
      init();
      this.contentPane = this;
      this.addCorrelatorPanelEventListener( paramEventListener );
   }
   catch ( Exception e )
   {
      e.printStackTrace();
   }
}   //end CorrelatorPanel constructor



public void addCorrelatorPanelEventListener( CorrelatorPanelEventListener
listener )
{
   this.listenerList.add( CorrelatorPanelEventListener.class, listener );
}


public void removeCorrelatorPanelEventListener( CorrelatorPanelEventListener
listener )
{
   this.listenerList.remove( CorrelatorPanelEventListener.class, listener );
}


public void fireCorrelatorPanelEvent( int eventID, int threshold,
String selectedName )
{

   //CCR[] ccrList = this.fiom.getUnregisteredCCR();

   //Object[] nameArray = new Object[ccrList.length];

   //for(int i=0; i < ccrList.length; i++)
   //{
   //    nameArray[i] = new String( ccrList[i].getCCRName() );
   //System.out.println(ccrList[i].getCCRName());
   //}//end for loop

   //Guaranteed to return a non-null array
   Object[] listeners = this.listenerList.getListenerList();
   // Process the listeners last to first, notifying
   // those that are interested in this event
   for ( int i = listeners.length - 2; i >= 0; i -= 2 )
   {
      CorrelatorPanelEvent event = null;
      if ( listeners[i] == CorrelatorPanelEventListener.class )
      {
         // Lazily create the event:
         if ( event == null )
         {
            event = new CorrelatorPanelEvent( this,
            eventID,
            threshold,
            selectedName );
         }
         ( ( CorrelatorPanelEventListener )listeners[i+1]).panelEventOccured(event);
      }
   }// end of for loop to notify listeners
}   //end fireCorrelatorPanelEvent()



private void init()
{

   this.setLayout( new BorderLayout() );
```

```
this.controlArea.setLayout( new FlowLayout() );

this.initCCRSelector();
//this.ccrSelectionSubPanel.setLayout(new GridLayout(2,1));
//this.ccrSelectionLabel.setText("Compoenent Class Selected:");
//this.ccrSelectionSubPanel.add(this.ccrSelectionLabel);
//this.ccrSelectionSubPanel.add(this.ccrSelected);


this.preferencesButton.setText( "Preferences" );
this.preferencesButton.setMargin( buttonInsets );
this.preferencesButton.setEnabled( true );
this.preferencesButton.addActionListener(
new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      preferencesButton_actionPerformed( e );
   }
} );



this.filterOnCombined.setText( "Combined Score" );
this.filterOnCombined.setMargin( buttonInsets );
this.filterOnCombined.setEnabled( true );
this.filterOnCombined.addActionListener(
new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      filterUsingCombined_actionPerformed( e );
   }
} );



this.filterOnKeyword.setText( "Keywords" );
this.filterOnKeyword.setMargin( buttonInsets );
this.filterOnKeyword.setEnabled( true );
this.filterOnKeyword.addActionListener(
new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      filterUsingKeywords_actionPerformed( e );
   }
} );


this.filterOnNeuralNet.setText( "Neural Net" );
this.filterOnNeuralNet.setMargin( buttonInsets );
this.filterOnNeuralNet.setEnabled( true );
this.filterOnNeuralNet.addActionListener(
new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      filterUsingNeuralNet_actionPerformed( e );
   }
} );



//get an array of proper length for Threahold values
this.thresholdValues = new Object[NUMBER_OF_VALUES];

//fill the array with values based on INCREMENT
```

```
for ( int i = 0; i < NUMBER_OF_VALUES; i++ )
{
   //I add one to i, so that the first element is not zero.
   this.thresholdValues[i] = String.valueOf( ( i + 1 ) * INCREMENT );
}

//then just build the combo boxes out of the array from above
this.keywordThreshold = new JComboBox( this.thresholdValues );
this.keywordThreshold.setSelectedIndex( DEFAULT_THRESHOLD_INDEX );

this.combinedScoreThreshold = new JComboBox( this.thresholdValues );
this.combinedScoreThreshold.setSelectedIndex( DEFAULT_THRESHOLD_INDEX );

this.neuralNetThreshold = new JComboBox( this.thresholdValues );
this.neuralNetThreshold.setSelectedIndex( DEFAULT_THRESHOLD_INDEX );


this.buttonLabel.setText( new String( "Filter On:" ) );
this.buttonLabel.setHorizontalAlignment( JLabel.RIGHT );
this.comboBoxLabel.setText( new String( "Threshold Values:" ) );
this.comboBoxLabel.setHorizontalAlignment( JLabel.RIGHT );


// set up the threshold buttons and combo boxes
this.thresholdSubPanel.setLayout( new GridLayout( 2, 4 ) );
this.thresholdSubPanel.add( this.buttonLabel );
this.thresholdSubPanel.add( this.filterOnKeyword );
this.thresholdSubPanel.add( this.filterOnNeuralNet );
this.thresholdSubPanel.add( this.filterOnCombined );
this.thresholdSubPanel.add( this.comboBoxLabel );
this.thresholdSubPanel.add( this.keywordThreshold );
this.thresholdSubPanel.add( this.neuralNetThreshold );
this.thresholdSubPanel.add( this.combinedScoreThreshold );


this.showSelectedFE.setText( new String( "Show FE" ) );
this.showSelectedFE.setMargin( buttonInsets );
this.showSelectedFE.setEnabled( true );
this.showSelectedFE.addActionListener(
new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      showSelectedFE_actionPerformed( e );
   }
} );



this.actionSubPanel.add( this.showSelectedFE );




//put all the sub comonents into the controlArea
this.controlArea.add( ccrSelectionSubPanel );
this.controlArea.add( this.preferencesButton );
this.controlArea.add( this.thresholdSubPanel );
this.controlArea.add( this.actionSubPanel );




  /* This was my first approach.  Much easier to just define the
    table model, as shown below.
  this.fevColumn.setHeaderValue(new String("FEV"));
  this.keywordScoreColumn.setHeaderValue(new String("Keyword Score"));
  this.passThruOptionColumn.setHeaderValue(new String("Pass-Thru"));
```

```
        this.neuralNetScoreColumn.setHeaderValue(new String("Neural Net Score"));
        this.combinedScoreColumn.setHeaderValue(new String("Combined Score"));
        this.selectFEColumn.setHeaderValue(new String("Select FE"));
        */


      //This sets up a table with 5 rows with the column headers found at
      //  the start of this class.
      TableModel dataModel = new AbstractTableModel()
      {
         public int getColumnCount() { return 6; }

         //This method is used to construct the table, and so
         //  by returning the desired heading label, the table is
         //  constructed each time to look as we desire.
         public String getColumnName( int columnIndex )
         {
            String columnName = null;
            switch ( columnIndex )
            {
               case( 0 ) :
                  columnName = CorrelatorPanel.COLUMN_ZERO_TITLE;
                  break;
               case( 1 ) :
                  columnName = CorrelatorPanel.COLUMN_ONE_TITLE;
                  break;
               case( 2 ) :
                  columnName = CorrelatorPanel.COLUMN_TWO_TITLE;
                  break;
               case( 3 ) :
                  columnName = CorrelatorPanel.COLUMN_THREE_TITLE;
                  break;
               case( 4 ) :
                  columnName = CorrelatorPanel.COLUMN_FOUR_TITLE;
                  break;
               case( 5 ) :
                  columnName = CorrelatorPanel.COLUMN_FIVE_TITLE;
                  break;
               default:
                  columnName = new String( "error" );
            }
            return ( columnName );
         }

         public Class getColumnClass( int column )
         {
            Class classValue = null;
            switch ( column )
            {
               //case(2) : case(5) : classValue = (Class)
javax.swing.JToggleButton.class;
               //break;
               default:
                  classValue = ( Class ) java.lang.Object.class;
            }
            return ( classValue );
         }


         public int getRowCount() { return 5; }

         //eventually this will need to be writen to link the table that we
         //  view, which is being defined here, to arrays of information that
         //  are passed into the CorrelatorPanel from the correlator.
         // So that as the table is constructed, this statement can
         //  return that which is to be displayed.
         //    right now I'm trying to display a blank table with
         //    checkboxes in the third and fifth columns.
         public Object getValueAt( int row, int col )
```

```
      {
         Object cellValue = null;
         switch ( col )
         {
            //case(2) : case(5) : cellValue = new JCheckBox();
            //   break;
            default:
               cellValue = new String( "" );
         }
         return ( cellValue );
      }
   };

   this.table = new JTable( dataModel );

   //this.add(table, BorderLayout.CENTER);
   dataArea = new JScrollPane( table );


   //put the control area and the data area in the main panel
   //this.add(this.controlArea, BorderLayout.NORTH);
   //this.add(this.dataArea, BorderLayout.CENTER);
   this.refreshCorrelatorPanel();

}   //end init


public void initCCRSelector()
{
   this.ccrSelected.addActionListener(
   new ActionListener()
   {
      public void actionPerformed( ActionEvent e )
      {
         changeToSelectedCCR( e );
      }
   } );

   this.ccrSelectionSubPanel = new JPanel();
   this.ccrSelectionSubPanel.setLayout( new GridLayout( 2, 1 ) );
   this.ccrSelectionLabel.setText( "Compoenent Class Selected:" );
   this.ccrSelectionSubPanel.add( this.ccrSelectionLabel );
   this.ccrSelectionSubPanel.add( this.ccrSelected );

}


/** ***************************************************************************
 * Most likely this will either accept a 2D array of
 * data that will be put in this panel's table for
 * display.
 *
 * The alternative is to construct the table else
 * where and send in the whole table.  It seems better
 * to just send in the data and let the panel
 * handle the table details.
 *
 *
 * Modified on 9/21/2002 by LT Steve Shedd
 *
 ***************************************************************************** */
public void setContent( Object[] [] tableContent )
{

   String[] columnNames =   { CorrelatorPanel.COLUMN_ZERO_TITLE,
                              CorrelatorPanel.COLUMN_ONE_TITLE,
                              CorrelatorPanel.COLUMN_TWO_TITLE,
                              CorrelatorPanel.COLUMN_THREE_TITLE,
                              CorrelatorPanel.COLUMN_FOUR_TITLE,
```

```
                              CorrelatorPanel.COLUMN_FIVE_TITLE };

   TableModel newModel = new DefaultTableModel( tableContent, columnNames );

   this.table = null;

   this.table = new JTable( newModel );
   this.dataArea = new JScrollPane( table );


   this.refreshCorrelatorPanel();


}   //end setContent



public void setCCRList( Object[] paramList )
{
   //Object[] tempList = this.parentFrame.controlChannel.getAvailableCCRList();

   //for(int i=0; i<tempList.length; i++)
   //{
   //    this.ccrSelected.addItem( tempList[i] );
   //}

   this.ccrSelected.removeAllItems();     // = new JComboBox();

   for ( int i = 0; i < paramList.length; i++ )
   {
      this.ccrSelected.addItem( paramList[i] );
      //System.out.println(paramList[i]);
   }

   //this.initCCRSelector();
   //System.out.println(this.ccrSelected.getItemCount());

   refreshCorrelatorPanel();
}   //end setCCRList


public void refreshCorrelatorPanel()
{
   this.removeAll();

   this.add( this.controlArea, BorderLayout.NORTH );
   this.add( this.dataArea, BorderLayout.CENTER );

   //this.contentPane = this;
}



// *************** Button Handlers ************************************

private void preferencesButton_actionPerformed( ActionEvent e )
{
   fireCorrelatorPanelEvent( CorrelatorPanelEvent.PREFERENCES_BUTTON_PRESSED,
   CorrelatorPanelEvent.NULL_INT,
   CorrelatorPanelEvent.NULL_STRING );
}   //end preferencesButton_actionPerformed



private void filterUsingKeywords_actionPerformed( ActionEvent e )
{
   int threshold = Integer.parseInt(
   ( this.keywordThreshold.getSelectedItem() ).toString() );
   fireCorrelatorPanelEvent( CorrelatorPanelEvent.KEYWORD_BUTTON_PRESSED,
```

```
      threshold,
      CorrelatorPanelEvent.NULL_STRING );
   }   //end filterUsingKeywords_actionPerformed



   private void filterUsingNeuralNet_actionPerformed( ActionEvent e )
   {
      int threshold = Integer.parseInt( ( this.neuralNetThreshold.getSelectedItem()
          ).toString() );
      fireCorrelatorPanelEvent( CorrelatorPanelEvent.NEURAL_NET_BUTTON_PRESSED,
      threshold,
      CorrelatorPanelEvent.NULL_STRING );
   }   //end filterUsingNeuralNet_actionPerformed



   private void filterUsingCombined_actionPerformed( ActionEvent e )
   {
      int threshold = Integer.parseInt( ( this.combinedScoreThreshold.getSelectedItem()
          ).toString() );
      fireCorrelatorPanelEvent( CorrelatorPanelEvent.COMBINED_BUTTON_PRESSED,
      threshold,
      CorrelatorPanelEvent.NULL_STRING );
   }   //end filterUsingCombined_actionPerformed



   private void showSelectedFE_actionPerformed( ActionEvent e )
   {
      try
      {
         //this.parentFrame.controlChannel.correlatorShowSelectedFE();
      }
      catch ( java.lang.UnsupportedOperationException uoe )
      {
         //this.parentFrame.notImplementedDialog( uoe );
      }
   }   //end showSelectedFE_actionPerformed


   private void changeToSelectedCCR( ActionEvent e )
   {
      fireCorrelatorPanelEvent( CorrelatorPanelEvent.CCR_SELECTED,
      CorrelatorPanelEvent.NULL_INT,
      ( ( String )this.ccrSelected.getSelectedItem() ) );
   }


   public JPanel getCorrelatorPanel()
   {
      return ( this.contentPane );
   }

}      //end CorrelatorPanel
```

F. PACKAGE: mil.navy.nps.cs.babel.connectors

1. Interface CMCorrelatorInterface

```
package mil.navy.nps.cs.babel.connectors;

import java.util.*;
import javax.swing.JComponent;

import org.jdom.Document;
```

```
/** Document the purpose of this class.
 *
 * @version 1.0
 * @author  LT Steve Shedd
 */

public interface CMCorrelatorInterface
{


    public void updateCCRList();

    public JComponent getCorrelatorPanel();

    public void generateCCRSemanticComponents(Document paramJDOMDoc);

    public void generateCCRSyntacticComponents(Document paramJDOMDoc);

    public void generateFCRSemanticComponents(Document paramJDOMDoc);

    public void generateFCRSyntacticComponents(Document paramJDOMDoc);

    public void doSyntacticCorrelation();

    public void doSemanticCorrelation();
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. FIOM FRAMEWORK MODIFICATION SOURCE

A. PACKAGE: mil.navy.nps.cs.oomi.fiom

1. Interface Semantics

```
//*****************************************************************************
// Filename:.............Semantics.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;


public interface Semantics
{

   /** ************************************************************************
    * Get the Keyword list held by this FEV object
    *
    * @author Steve Shedd
    ************************************************************************** */
   public String[] getKeywordList();


   /** ************************************************************************
    * Sets a keyword list for an FEV object.
    *
    * @author Steve Shedd
    ************************************************************************** */
   public void setKeywordList(String[] in);


}
```

2. Interface Syntax

```
//*****************************************************************************
// Filename:.............Syntax.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;

import java.util.List;


public interface Syntax
{

   /** ************************************************************************
    * Get the discriminator vectors for this FEV object
    *
    * @author Steve Shedd
    ************************************************************************** */
```

```
   public List getDiscriminatorVectors();


   /** ****************************************************************************
    * Set discriminator vectors for this FEV object
    *
    * @author Steve Shedd
    ***************************************************************************** */
   public void setDiscriminatorVectors(List in);


}
```

3. Interface CCR

```
//*****************************************************************************
// Filename:.............CCR.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............Lee Shong Cheng [Lee02]
// Modified By:..........LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/** ****************************************************************************
 * Represents a CCR in OOMI.
 * A CCR should be uniquely defined by:
 *  <ol>
 *    <li> SystemName and CCRName
 *    <li> SystemName and XMLNameSpaceURI
 *  </ol>
 * within the FIOM.
 * <p>
 * Responsibilities:
 * <ul>
 * <li> Represents a CCR in the OOMI FIOM.
 * <li> Maintain information on whether an attribute of its FCR is mandatory,
 *      based on the FCRtoCCR attribute mappings contained in this object.
 * </ul>
 *
 * @author LSC
 * @version 1.0
 ***************************************************************************** */

public interface CCR
{

   /** Returns the name of this CCR. */
   public String getCCRName();


   /** Returns the name of the system this CCR is defined for. */
   public String getSystemName();


   /** Returns the FCR of this CCR. */
   public FCR getFCR() ;
```

```
/** ****************************************************************************
 * Joins this CCR to the specified FCR.
 * If bewFCR==null, nothing is done.
 * @param newFCR The FCR that this CCR will be joining.
 ***************************************************************************** */
public void joinFCR( FCR newFCR ) ;


/** Returns the CCRSchema for this object. */
public CCRSchema getCCRSchema();


/** ****************************************************************************
 * Returns the fully-qualified Castor
 * generated Java class's (a descendent of CCRInstance) name for this CCR.
 * A String is used instead of Class so that it may be possible
 * to work with the CCR without actually loading the bytecode for the
 * class.
 ***************************************************************************** */
public String getJavaClassName();


/** ****************************************************************************
 * Sets the fully-qualified name of the Castor
 * generated Java class (a descendent of CCRInstance) for this CCR.
 ***************************************************************************** */
public void setJavaClassName( String javaClassName );


/** Returns the XML Schema URI this CCR is defined for. */
public String getXMLNameSpaceURI();


/** ****************************************************************************
 * Sets the XML Schema URI this CCR is defined for.
 * A well-formed valid URI is expected, no error checking is performed.
 ***************************************************************************** */
public void setXMLNameSpaceURI( String xmlNameSpaceURI );


/** ****************************************************************************
 * The given FCR attribute is mandatory if it is mapped to one or more mandatory
 * CCR attributes.
 *
 ***************************************************************************** */
public boolean isMandatoryFCRAttribute( Attribute fcrAttr );


/** ****************************************************************************
 * Returns an array of mandatory attributes of the FCR this CCR is associated
 * with.
 ***************************************************************************** */
public Attribute[] mandatoryFCRAttributes();


/** ****************************************************************************
 * Returns the number of mandatory attributes of the FCR this CCR is associated
 * with.
 ***************************************************************************** */
public int mandatoryFCRAttributeCount();


/** Adds a mapping of FCR attributes to the attribute of this CCR object. */
public void addFCRToCCRAttributeMapping( Attribute[] from, Attribute to );


/** Clears the FCR to CCR attribute mappings. */
public void clearFCRToCCRAttributeMappings();
```

```
   /** ****************************************************************************
    * Get the CCR Semantics File
    *
    * @author Steve Shedd
    **************************************************************************** */
   public CCRSemantics getCCRSemantics();


   /** ****************************************************************************
    * Sets the semantics for this CCR.
    *
    * @author Steve Shedd
    **************************************************************************** */
   public void setCCRSemantics( CCRSemantics sem );


   /** ****************************************************************************
    * Get the CCR Syntax File
    *
    * @author Steve Shedd
    **************************************************************************** */
   public CCRSyntax getCCRSyntax();


   /** ****************************************************************************
    * Sets the syntax for this CCR.
    *
    * @author Steve Shedd
    **************************************************************************** */
   public void setCCRSyntax( CCRSyntax syn );

}
```

4. Interface CCRSemantics

```
//*****************************************************************************
// Filename:.............CCRSemantics.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;


public interface CCRSemantics extends Semantics
{

   // Stubbed out for future extensions

}
```

5. Interface CCRSyntax

```
//*****************************************************************************
// Filename:.............CCRSyntax.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************
```

```
package mil.navy.nps.cs.oomi.fiom;


public interface CCRSyntax extends Syntax
{

   // Stubbed out for future extensions

}
```

6. Interface FCR

```
//*****************************************************************************
// Filename:.............FCR.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............Lee Shong Cheng [Lee02]
// Modified By:..........LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;

import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/** ***************************************************************************
 * @author LSC
 * @version 1.0
 **************************************************************************** */

public interface FCR
{

   /** Returns the name of this FCR. */
   public String getFCRName();


   /** Returns the FEV of this FCR. */
   public FEV getFEV();


   /** Returns the number of CCRs defined for this FCR. */
   public int ccrCount();


   /** ***************************************************************************
    * Returns all the CCRs defined for this FCR in an array.
    * @return Returns all the CCRs defined for this FCR in an array,
    *         an empty array is returned if no CCRs defined.
    **************************************************************************** */
   public CCR[] getCCR();


   /** ***************************************************************************
    * Finds all the CCRs which satisifies the specified criteria.
    * @param criteria
    **************************************************************************** */
   public void findCCR( SearchHandler handler );
```

```
/** ****************************************************************************
 * Finds all the CCR with for the specified system.
 * @param systemName
 * @return Returns all the CCRs in this FCR that belongs to the
 * specified system name
 *         (not case-sensitive).
 *         Returns an empty array if not found.
 ***************************************************************************** */
public CCR[] findCCR( String systemName );


/** ****************************************************************************
 * Finds the CCR with the specified name.
 * @param systemName
 * @param ccrName
 * @return Returns the CCR in this FCR that has the specified name
 *         (not case-sensitive).
 *         Returns null if not found.
 ***************************************************************************** */
public CCR findCCR( String systemName, String ccrName );


/** ****************************************************************************
 * Finds the CCR with the specified system name and xmlNameSpaceURI.
 * @param systemName
 * @param xmlNameSpaceURI
 * @return Returns the CCR in this FCR that has the specified system name
 *         (not case-sensitive) and xmlNameSpaceURI.
 *         Returns null if not found.
 *         xmlNameSpaceURI==null shall always result in a return value of null.
 ***************************************************************************** */
public CCR findCCR( String systemName, URL xmlNameSpaceURI );


/** Returns true if this FCR has a CCR with specified name. */
public boolean ccrExists( String systemName, String ccrName );


/** ****************************************************************************
 * Returns true if this FCR has a CCR with specified name.
 * @return Returns true if this FCR has a CCR with specified name.
 *         Returns false if xmlNameSpaceURI==null.
 ***************************************************************************** */
public boolean ccrExists( String systemName, URL xmlNameSpaceURI );


/** ****************************************************************************
 * Creates a new instance of CCR for this FCR and returns the newly created
 * instance.
 * @param systemName The system name.
 * @param ccrName The name of the new CCR.
 * @param xmlNameSpaceURI The URL to the XML schema that the new CCR represents.
 *                     Pass in null if XML schema is not required.
 *
 * @throws DuplicateKeyException When the specified systemName,ccrName pair or
 *                               systemName, xmlNameSpaceURI pair
 *                               already exist for this FCR.
 ***************************************************************************** */
public CCR newCCR( String systemName, String ccrName, URL xmlNameSpaceURI )
throws DuplicateKeyException;


/** Returns the FCRSchema for this object. */
public FCRSchema getFCRSchema();


/** Returns the Java class name for this FCR. */
public String getJavaClassName();
```

```
   /** Sets the Java class name for this FCR. */
   public void setJavaClassName( String className );


   /** ****************************************************************************
    * Finds the CCR that represents the specified XML schema.
    * A case-sensitive comparison is performed.
    * @param xmlNameSpaceURI The URL to the XML schema whose corresponding CCR
    *                        is to be found.
    * @return Returns the CCR that represents the specified XML schema
    *         corresponding to this FCR.
    *         Returns null if no such CCR exists or if xmlNameSpaceURI==null.
    ******************************************************************************* */
   //  public CCR findCCR( URL xmlNameSpaceURI );


   /** ****************************************************************************
    * Get the FCR Semantics File
    *
    * @author Steve Shedd
    ******************************************************************************* */
   public FCRSemantics getFCRSemantics();


   /** ****************************************************************************
    * Sets the semantics for this FCR.
    *
    * @author Steve Shedd
    ******************************************************************************* */
   public void setFCRSemantics( FCRSemantics sem );


   /** ****************************************************************************
    * Get the FCR Syntax File
    *
    * @author Steve Shedd
    ******************************************************************************* */
   public FCRSyntax getFCRSyntax();


   /** ****************************************************************************
    * Sets the syntax for this FCR.
    *
    * @author Steve Shedd
    ******************************************************************************* */
   public void setFCRSyntax( FCRSyntax syn );

}
```

7. Interface FCRSemantics

```
//*****************************************************************************
// Filename:.............FCRSemantics.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;


public interface FCRSemantics extends Semantics
{
```

```
   // Stubbed out for future additions

}
```

8. Interface FCRSyntax

```
//*****************************************************************************
// Filename:.............FCRSyntax.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.fiom;


import mil.navy.nps.cs.oomi.fiom.*;
import mwa.ai.neural.*;


public interface FCRSyntax extends Syntax
{

   /** ************************************************************************
    * Get neural network for this FCR.
    *
    * @author Steve Shedd
    ************************************************************************** */
   public Neural getNeuralNetwork();

   /** ************************************************************************
    * Get neural network for this FCR.
    *
    * @author Steve Shedd
    ************************************************************************** */
   public void setNeuralNetwork( Neural netIn );

   /** ************************************************************************
    * Get the path to the neural network file for this FCR.
    *
    * @author Steve Shedd
    ************************************************************************** */
   public String getNNfilePath();

   /** ************************************************************************
    * Set the path for the neural network file for this FCR.
    *
    * @author Steve Shedd
    ************************************************************************** */
   public void setNNfilePath( String absolutePath );


}
```

B. PACKAGE: mil.navy.nps.cs.oomi.impl

1. Class SemanticsImpl

```
//*****************************************************************************
// Filename:.............SemanticsImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
```

```
// Company:.............Naval Postgraduate School, Monterey, California
// Date:................September 2002
//*****************************************************************************


package mil.navy.nps.cs.oomi.impl;

import java.io.Serializable;
import mil.navy.nps.cs.oomi.fiom.Semantics;


public abstract class SemanticsImpl implements Semantics, Serializable
{

   protected final static int MAX_KEYWORD_LIST_SIZE = 10000;

   protected String[] keywordList = new String[MAX_KEYWORD_LIST_SIZE];


   public SemanticsImpl()
   {

   }


   /** ************************************************************************
    * Get the Keyword list held by this semantics object
    *
    * @author Steve Shedd
    ************************************************************************** */
   public String[] getKeywordList()
   {
      return keywordList;
   }


   /** ************************************************************************
    * Set the keyword list for an FEV object.
    *
    * @author Steve Shedd
    ************************************************************************** */
   public void setKeywordList(String[] in)
   {
      if ( in.length > 10000 )
      {
         for ( int i = 0; i < keywordList.length; i++ )
         {
            keywordList[i] = in[i];
         }
      }
      else
      {
         keywordList = in;
      }

   }

}
```

2. Class SyntaxImpl

```
//*****************************************************************************
// Filename:............SyntaxImpl.java
// Parent Project:.....SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                     Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:..............LT Steve Shedd, USN
// Company:.............Naval Postgraduate School, Monterey, California
```

```
// Date:.................September 2002
//*****************************************************************************


package mil.navy.nps.cs.oomi.impl;

import java.io.Serializable;
import java.util.List;
import java.util.ArrayList;
import mil.navy.nps.cs.oomi.fiom.Syntax;


public abstract class SyntaxImpl implements Syntax, Serializable
{

   List _discriminatorVectors = new ArrayList();


   public SyntaxImpl()
   {

   }


   /** ************************************************************************
    * Get the discriminator vectors for this FEV object
    *
    * @author Steve Shedd
    ************************************************************************** */
   public List getDiscriminatorVectors()
   {
      return _discriminatorVectors;
   }


   /** ************************************************************************
    * Set discriminator vectors to this FEV object
    *
    * @author Steve Shedd
    ************************************************************************** */
   public void setDiscriminatorVectors( List in )
   {
      _discriminatorVectors.addAll( in );
   }

}
```

3. Class CCRImpl

```
//*****************************************************************************
// Filename:.............CCRImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............Lee Shong Cheng [Lee02]
// Modified By:..........LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.impl;

import java.util.*;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.impl.*;
import mil.navy.nps.cs.oomi.exceptions.*;
```

```
/** ****************************************************************************
 * A reference implementation of the Component Class Representation (CCR).
 * Needs to maintain the mandatory attributes of the corresponding FCR.
 *
 * @author LSC
 * @version 1.0
 ******************************************************************************* */

public class CCRImpl implements CCR, Comparable
{

   /** FCR to CCR attribute mappings. */
   protected List _fcrToCcrAttrMappingList = new ArrayList();


   /** The system name of this CCR. */
   protected String _systemName = "";


   /** The name of this CCR. */
   protected String _ccrName = "";


   /** The FCR corresponding to this CCR. */
   protected FCR _fcr = null;


   /** The schema for this CCR. */
   protected CCRSchema _ccrSchema = new CCRSchemaImpl();


   /** ****************************************************************************
    * The semantics for this CCR.
    *
    * @author Steve Shedd
    ******************************************************************************* */
   protected CCRSemantics _ccrSemantics = new CCRSemanticsImpl();


   /** ****************************************************************************
    * The syntax for this CCR.
    *
    * @author Steve Shedd
    ******************************************************************************* */
   protected CCRSyntax _ccrSyntax = new CCRSyntaxImpl();


   /** The fully qualified name of the Castor generated class for the XML schema. */
   protected String _javaClassName = "";


   /** The URL to the XML schema for which this CCR is defined for. */
   protected String _xmlNameSpaceURI = "";


   /** ****************************************************************************
    * An array of mandatory FCR attributes.
    * Transient field, built from _fcrToCcrAttrMappingList when required.
    ******************************************************************************* */
   protected Set _mandatoryFCRAttributes = new HashSet();


   /** Constructor required by Castor Unmarshaller. */
   public CCRImpl()
   {
   }
```

```
/** ****************************************************************************
 * Initializes the CCR.
 * If the specified FCR is not null, this CCR shall join the specified FCR.
 * @param fcr         The FCR this CCR will join.
 * @param systemName  System name of this CCR.
 * @param ccrName     Name of this CCR.
 ***************************************************************************** */
public CCRImpl( FCR fcr, String systemName, String ccrName )
{
   joinFCR( fcr );
   setCCRName( ccrName );
   setSystemName( systemName );
}


/** ****************************************************************************
 * Sets the name of this CCR.
 * Public access required by Castor Unmarshaller.
 ***************************************************************************** */
public void setCCRName( String ccrName )
{
   _ccrName = ccrName;
}


public String getCCRName()
{
   return _ccrName;
}


/** ****************************************************************************
 * Sets the FCR of this object.
 * Used only for Castor Marshaling/Unmarshaling.
 ***************************************************************************** */
void setFCR( FCR fcr )
{
   _fcr = fcr;
}


public FCR getFCR()
{
   return _fcr;
}


public void joinFCR( FCR newFCR )
{
   if ( newFCR != null )
   {
      ( ( FCRImpl ) newFCR ).addCCR( this );
   }
}


public CCRSchema getCCRSchema()
{
   return _ccrSchema;
}


/** Public access required by Castor Unmarshaller. */
public void setCCRSchema( CCRSchema v )
{
   _ccrSchema = v;
}
```

```
/** ***************************************************************************
 * Gets the semantics for this CCR.
 *
 * @author Steve Shedd
 **************************************************************************** */
public CCRSemantics getCCRSemantics()
{
   return _ccrSemantics;
}


/** ***************************************************************************
 * Sets the semantics for this CCR.
 *
 * @author Steve Shedd
 **************************************************************************** */
public void setCCRSemantics( CCRSemantics sem )
{
   _ccrSemantics = sem;
}


/** ***************************************************************************
 * Gets the syntax for this CCR.
 *
 * @author Steve Shedd
 **************************************************************************** */
public CCRSyntax getCCRSyntax()
{
   return _ccrSyntax;
}


/** ***************************************************************************
 * Sets the syntax for this CCR.
 *
 * @author Steve Shedd
 **************************************************************************** */
public void setCCRSyntax( CCRSyntax syn )
{
   _ccrSyntax = syn;
}


public String getJavaClassName()
{
   return _javaClassName;
}


public void setJavaClassName( String javaClassName )
{
   _javaClassName = javaClassName;
}


public String getXMLNameSpaceURI()
{
   return _xmlNameSpaceURI;
}


public void setXMLNameSpaceURI( String xmlNameSpaceURI )
{
   _xmlNameSpaceURI = xmlNameSpaceURI;
}
```

```
public String getSystemName()
{
   return _systemName;
}


public void setSystemName( String v )
{
   _systemName = v;
}


public AttributeMapping[] getFCRToCCRAttributeMapping()
{
   return ( AttributeMapping[] ) _fcrToCcrAttrMappingList.toArray(
   new AttributeMapping[_fcrToCcrAttrMappingList.size()] );
}


public void setFCRToCCRAttributeMapping( AttributeMapping[] v )
{
   clearFCRToCCRAttributeMappings();
   for ( int i = 0; i < v.length; i++ )
   {
      _fcrToCcrAttrMappingList.add( v[i] );
   }
}


public Attribute[] mandatoryFCRAttributes()
{
   return ( Attribute[] ) mandatoryFCRAttributeSet().toArray(
   new Attribute[mandatoryFCRAttributeCount()] );
}


public boolean isMandatoryFCRAttribute( Attribute fcrAttr )
{
   return mandatoryFCRAttributeSet().contains( fcrAttr );
}


public int mandatoryFCRAttributeCount()
{
   return mandatoryFCRAttributeSet().size();
}


/** ***************************************************************************
 * Returns the mandatory FCR attributes set, builds the set by calling
 * {@link #buildMandatoryFCRAttributeSet()} if the set is empty.
 ***************************************************************************** */
protected Set mandatoryFCRAttributeSet()
{
   if ( _mandatoryFCRAttributes.size() == 0 )
   { buildMandatoryFCRAttributeSet(); }
   return _mandatoryFCRAttributes;
}


/** ***************************************************************************
 * Builds the mandatory FCR attributes set, based on the current
 * _fcrToCcrAttrMappingList.
 ***************************************************************************** */
protected void buildMandatoryFCRAttributeSet()
{
   // first, clear the existing set
   _mandatoryFCRAttributes.clear();
   CCRSchema ccrSchema = getCCRSchema();
```

```
      Iterator iter = _fcrToCcrAttrMappingList.iterator();
      while ( iter.hasNext() )
      {
         AttributeMapping mapping = ( AttributeMapping )iter.next();
         Attribute[] fcrAttrs     = mapping.fromAttributes();
         Attribute   ccrAttr      = mapping.toAttribute();
         if ( ccrSchema.findAttribute( ccrAttr ).isMandatory() )
         {
            for ( int i = 0; i < fcrAttrs.length; i++ )
            {
               _mandatoryFCRAttributes.add( fcrAttrs[i] );
            }
         }
      }
   }


   public void addFCRToCCRAttributeMapping( Attribute[] from, Attribute to )
   {
      AttributeMapping mapping = FactoryImpl.factory().makeAttributeMapping( from, to );
      _fcrToCcrAttrMappingList.add( mapping );
   }


   public void clearFCRToCCRAttributeMappings()
   {
      _fcrToCcrAttrMappingList.clear();
   }


   /** ****************************************************************************
    * Used mainly for sorting required during testing.
    * Ordering is based on SystemName order followed by CCRName.
    **************************************************************************** */
   public int compareTo( Object o )
   {
      CCR ccr = ( CCR ) o;
      if ( this == o )
      { return 0; }
      int c;
      c = getSystemName().compareTo( ccr.getSystemName() );
      if ( c != 0 )
      { return c; }
      else { return getCCRName().compareTo( ccr.getCCRName() ); }
   }


   static void p( String m )
   {
      System.out.println( m );
   }
}
```

4. Class CCRSemanticsImpl

```
//*****************************************************************************
// Filename:.............CCRSemanticsImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.fiom.CCRSemantics;
```

```
public class CCRSemanticsImpl extends SemanticsImpl implements CCRSemantics
{

   public CCRSemanticsImpl()
   {
      super();
   }

}
```

5. Class CCRSyntaxImpl

```
//*****************************************************************************
// Filename:.............CCRSyntaxImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************

package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.fiom.CCRSyntax;


public class CCRSyntaxImpl extends SyntaxImpl implements CCRSyntax
{

   public CCRSyntaxImpl()
   {
      super();
   }

}
```

6. Class FCRImpl

```
//*****************************************************************************
// Filename:.............FCRImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............Lee Shong Cheng [Lee02]
// Modified By:..........LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*****************************************************************************


package mil.navy.nps.cs.oomi.impl;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.HashSet;
import java.util.Arrays;
import java.util.Collections;
import java.util.Collection;
import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
```

```
import mil.navy.nps.cs.oomi.impl.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/** ****************************************************************************
 *
 * @author LSC
 * @version 1.0
 ***************************************************************************** */

public class FCRImpl implements FCR
{

   /** The name of this FCR. */
   protected String _fcrName = "";


   /** The Java class name that implements the FCRInstance for this FCR. */
   protected String _javaClassName;


   /** The FEV that corresponds to this FCR. */
   protected FEV _fev = null;


   /** The list of CCRs corresponding to this FCR. */
   protected Set _ccrSet = new HashSet();


   /** The schema for this FCR. */
   protected FCRSchema _fcrSchema = new FCRSchemaImpl();


   /** ****************************************************************************
    * The semantics for this FCR.
    *
    * @author Steve Shedd
    ***************************************************************************** */
   protected FCRSemantics _fcrSemantics = new FCRSemanticsImpl();


   /** ****************************************************************************
    * The syntax for this FCR.
    *
    * @author Steve Shedd
    ***************************************************************************** */
   protected FCRSyntax _fcrSyntax = new FCRSyntaxImpl();



   /** Required for Castor Marshaller and Unmarshaller. */
   public FCRImpl()
   {

   }


   /** Creates a new instance with specified name. */
   public FCRImpl( FEV fev, String fcrName )
   {
      setFCRName( fcrName );
      setFEV( fev );
   }


   public String getFCRName()
   {
      return _fcrName;
   }
```

```
/** ****************************************************************************
 * Sets the name of this FCR.
 * To be used only by the constructor
 * and during restoration from persistent storage, using
 * Castor Unmarshaller.
 ******************************************************************************* */
public void setFCRName( String fcrName )
{
   _fcrName = fcrName;
}


/** ****************************************************************************
 * Sets the FEV of this FCR.
 * To be used only by the constructor
 * and by Castor Unmarshaller during restoration from persistent storage.
 *
 ******************************************************************************* */
public void setFEV( FEV value )
{
   _fev = value;
}


public CCR[] getCCR()
{
   return ( CCR[] ) _ccrSet.toArray( new CCR[ccrCount()] );
}


public FEV getFEV()
{
   return _fev;
}


public int ccrCount()
{
   return _ccrSet.size();
}


public void findCCR( SearchHandler handler )
{
   Iterator iter = _ccrSet.iterator();
   CCR ccr;
   while ( iter.hasNext() )
   {
      ccr = ( CCR )iter.next();
      if ( handler.satisfied( ccr ) )
      { handler.add( ccr ); }
   }
}


public CCR[] findCCR( String systemName )
{
   java.util.HashSet result = new java.util.HashSet();
   findCCR( makeCCRSystemNameSearchHandler( systemName, result ) );
   return ( CCR[] ) result.toArray( new CCR[result.size()] );

 /*
 List result = new ArrayList();
 Iterator iter = _ccrList.iterator();
 CCR ccr;
 while (iter.hasNext()) {
   ccr = (CCR)iter.next();
```

```
    if ( ccr.getSystemName().equalsIgnoreCase(systemName) )
      result.add(ccr);
  }
  return (CCR[]) result.toArray(new CCR[result.size()]);
  */
 }


 public CCR findCCR( String systemName, String ccrName )
 {
    Iterator iter = _ccrSet.iterator();
    CCR ccr;
    while ( iter.hasNext() )
    {
       ccr = ( CCR )iter.next();
       if ( ccr.getSystemName().equalsIgnoreCase( systemName ) &&
       ccr.getCCRName().equalsIgnoreCase( ccrName ) )
       { return ccr; }
    }
    return null;
 }


 public CCR findCCR( String systemName, URL xmlNameSpaceURI )
 {
    if ( xmlNameSpaceURI == null )
    { return null; }
    Iterator iter = _ccrSet.iterator();
    CCR ccr;
    String urlString = xmlNameSpaceURI.toString();
    while ( iter.hasNext() )
    {
       ccr = ( CCR )iter.next();
       if ( ccr.getSystemName().equalsIgnoreCase( systemName ) &&
       ccr.getXMLNameSpaceURI().equals( urlString ) )
       { return ccr; }
    }
    return null;
 }


/*
public CCR findCCR( URL xmlNameSpaceURI ) {
  if (xmlNameSpaceURI==null)
    return null;
  Iterator iter = _ccrList.iterator();
  CCR ccr;
  String urlString = xmlNameSpaceURI.toString();
  while (iter.hasNext()) {
    ccr = (CCR)iter.next();
    if ( ccr.getXMLNameSpaceURI().equals(urlString) )
      return ccr;
  }
  return null;
}
*/
 public boolean ccrExists( String systemName, String ccrName )
 {
    return findCCR( systemName, ccrName ) != null;
 }


 public boolean ccrExists( String systemName, URL xmlNameSpaceURI )
 {
    return findCCR( systemName, xmlNameSpaceURI ) != null;
 }


 public CCR newCCR( String systemName, String ccrName, URL xmlNameSpaceURI )
```

```
throws DuplicateKeyException
{
   if ( ccrExists( systemName, ccrName ) )
   {
      throw new DuplicateKeyException( ccrName +
          " already exists for system: " + systemName );
   }
   if ( ccrExists( systemName, xmlNameSpaceURI ) )
   {
      throw new DuplicateKeyException( xmlNameSpaceURI +
          " already exists for system: " + systemName );
   }
   CCRImpl ccr = new CCRImpl( this, systemName, ccrName );
   if ( xmlNameSpaceURI != null )
   { ccr.setXMLNameSpaceURI( xmlNameSpaceURI.toString() ); }
   _ccrSet.add( ccr );
   return ccr;
}


/** ************************************************************************
 * Adds the specified CCR object to this FCR,
 * <strong> the FCR field of the CCRImpl parameter is updated to
 * point to this object. </strong>
 * <p>Also used by Castor Unmarshaller only.
 *
 ************************************************************************** */
public void addCCR( CCR newCCR )
{
   ( ( CCRImpl )newCCR ).setFCR( this );
   _ccrSet.add( newCCR );
}


public FCRSchema getFCRSchema()
{
   return _fcrSchema;
}


/** Public access required by Castor Unmarshaller. */
public void setFCRSchema( FCRSchema v )
{
   _fcrSchema = v;
}


/** ************************************************************************
 * Gets the semantics for this FCR.
 *
 * @author Steve Shedd
 ************************************************************************** */
public FCRSemantics getFCRSemantics()
{
   return _fcrSemantics;
}


/** ************************************************************************
 * Sets the semantics for this FCR.
 *
 * @author Steve Shedd
 ************************************************************************** */
public void setFCRSemantics( FCRSemantics sem )
{
   _fcrSemantics = sem;
}
```

```
   /** ****************************************************************************
    * Gets the syntax for this FCR.
    *
    * @author Steve Shedd
    ***************************************************************************** */
   public FCRSyntax getFCRSyntax()
   {
      return _fcrSyntax;
   }


   /** ****************************************************************************
    * Sets the syntax for this FCR.
    *
    * @author Steve Shedd
    ***************************************************************************** */
   public void setFCRSyntax( FCRSyntax syn )
   {
      _fcrSyntax = syn;
   }


   public String getJavaClassName()
   {
      return _javaClassName;
   }


   public void setJavaClassName( String className )
   {
      _javaClassName = className;
   }


   /** ****************************************************************************
    * A factory method that returns an instance of SearchHandler for search of
    * CCRs by system name of the CCRs.
    ***************************************************************************** */
   public static SearchHandler makeCCRSystemNameSearchHandler(
   String systemName,
   java.util.Set resultSet )
   {
      return new CCRSystemNameSearchHandler( systemName, resultSet );
   }


   /** ****************************************************************************
    * @param systemName
    * @param xmlNamespaceURI
    * @param result A single element array to hold the result, the element holds
    *               a null value if no match found.
    ***************************************************************************** */
   public static CCRKeySearchHandler makeCCRKeySearchHandler(
   String systemName, URL xmlNamespaceURI,
   CCR[] result )
   {
      if ( result.length < 1 )
      {
         throw new IllegalArgumentException( "Argument result must not be an empty
array." );
      }
      return new CCRKeySearchHandler( systemName, xmlNamespaceURI, result );
   }


   static void p1( String m )
   {
      System.out.println( m );
   }
```

```
}
```

7. Class FCRSemanticsImpl

```
//*********************************************************************************
// Filename:.............FCRSemanticsImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*********************************************************************************


package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.fiom.FCRSemantics;


public class FCRSemanticsImpl extends SemanticsImpl implements FCRSemantics
{

   public FCRSemanticsImpl()
   {
      super();
   }
}
```

8. Class FCRSyntaxImpl

```
//*********************************************************************************
// Filename:.............FCRSyntaxImpl.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
//*********************************************************************************

package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.fiom.FCRSyntax;
import mwa.ai.neural.Neural;
import mwa.ai.neural.NNfile;


public class FCRSyntaxImpl extends SyntaxImpl implements FCRSyntax
{

   private Neural _neuralNet = null;

   private String _nnFilePath = null;


   public FCRSyntaxImpl()
   {

   }


   /** ***************************************************************************
    * Get neural network for this FCR.
    *
    * @author Steve Shedd
    ******************************************************************************* */
```

```
   public Neural getNeuralNetwork()
   {
      return _neuralNet;
   }


   /** ****************************************************************************
    * Set neural network for this FCR.
    *
    * @author Steve Shedd
    **************************************************************************** */
   public void setNeuralNetwork( Neural netIn )
   {
      _neuralNet = netIn;
   }


   /** ****************************************************************************
    * Get the path to the neural network file for this FCR.
    *
    * @author Steve Shedd
    **************************************************************************** */
   public String getNNfilePath()
   {
      return _nnFilePath;
   }


   /** ****************************************************************************
    * Set the path for the neural network file for this FCR.
    *
    * @author Steve Shedd
    **************************************************************************** */
   public void setNNfilePath( String absolutePath )
   {
      _nnFilePath = absolutePath;
   }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. NEURAL NETWORK SOURCE

A. CLASS mwa.ai.neural.Neural

```
//*****************************************************************************
// Filename:.............Neural.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............Mark Watson, in "Intelligent Java Applications"
// Date Created:.........8/5/1996
// Modified by:..........LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
// Note:.................File modified for three reasons:
//
//         (1) To confrom to JDK 1.3.1 and deprecated methods
//         (2) To improve readability (improved commenting and layout)
//         (3) Needs of thesis work
//
// Specific comments are provided where important code has been changed from
// the original.  Deletions of minor code snippets, like System.out.println
// calls, are not commented.
//*****************************************************************************

package mwa.ai.neural;

import java.awt.*;
import java.applet.Applet;
import java.lang.*;
import java.util.*;


//import mwa.gui.GUI;
//import mwa.data.*;

public class Neural extends Object
{

   // For debug output:
   //GUI MyGUI = null;

   //Next line added by Steve Shedd
   public float learningRate;

   public int NumInputs;

   public int NumHidden;

   public int NumOutputs;

   public int NumTraining;

   protected int WeightsFlag;

   protected int SpecialFlag;

   public float Inputs[];

   //Next four lines added by Steve Shedd
   public float HiddenLayer_In[];

   public float HiddenLayer_Out[];

   public float OutputLayer_In[];
```

```
public float OutputLayer_Out[];

//public float Hidden[];
//public float Outputs[];

public float W1[] [];

public float W2[] [];

public float hidden_bias = 1.0f;

public float output_bias = 1.0f;

public float output_errors[];

public float hidden_errors[];

//Next two lines added by Steve Shedd
public float diff[];

public float avgNetError;

public float InputTraining[];

public float OutputTraining[];

// mask of training examples to ignore (true -> ignore):
public boolean IgnoreTraining[] = null;

// mask of Input neurons to ignore:
public boolean IgnoreInput[] = null;

public NNfile NeuralFile = null;


public Neural()
{
   NumInputs = NumHidden = NumOutputs = 0;
}


public Neural( String file_name )
{

   NeuralFile = new NNfile( file_name );
   NumInputs = NeuralFile.NumInput;
   NumHidden = NeuralFile.NumHidden;
   NumOutputs = NeuralFile.NumOutput;
   NumTraining = NeuralFile.NumTraining;
   WeightsFlag = NeuralFile.WeightFlag;
   SpecialFlag = NeuralFile.SpecialFlag;

   //Next line added by Steve Shedd
   learningRate = .25f;

   Inputs = new float[NumInputs];

   //Next four lines added by Steve Shedd
   HiddenLayer_In = new float[NumHidden];
   HiddenLayer_Out = new float[NumHidden];
   OutputLayer_In = new float[NumOutputs];
   OutputLayer_Out = new float[NumOutputs];

   //Hidden = new float[NumHidden];
   //Outputs = new float[NumOutputs];
   W1 = new float[NumInputs] [NumHidden];
   W2 = new float[NumHidden] [NumOutputs];

   avgNetError = 0;
```

```
   // Retrieve the weight values from the NNfile object:
   if ( WeightsFlag != 0 )
   {
      for ( int i = 0; i < NumInputs; i++ )
      {
         for ( int h = 0; h < NumHidden; h++ )
         {
            W1[i] [h] = NeuralFile.GetW1( i, h );
         }
      }

      for ( int h = 0; h < NumHidden; h++ )
      {
         for ( int o = 0; o < NumOutputs; o++ )
         {
            W2[h] [o] = NeuralFile.GetW2( h, o );
         }
      }

   }
   else
   {
      randomizeWeights();
   }

   //Next line added by Steve Shedd
   diff = new float[NumOutputs];

   output_errors = new float[NumOutputs];
   hidden_errors = new float[NumHidden];

   // Get the training cases (if any) from the training file:
   LoadTrainingCases();

}


public Neural( int i, int h, int o )
{
   System.out.println( "In BackProp constructor" );
   Inputs = new float[i];

   //Hidden = new float[h];
   //Outputs = new float[o];
   W1 = new float[i] [h];
   W2 = new float[h] [o];
   NumInputs = i;
   NumHidden = h;
   NumOutputs = o;
   output_errors = new float[NumOutputs];
   hidden_errors = new float[NumHidden];

   //Next four lines added by Steve Shedd
   HiddenLayer_In = new float[NumHidden];
   HiddenLayer_Out = new float[NumHidden];
   OutputLayer_In = new float[NumOutputs];
   OutputLayer_Out = new float[NumOutputs];

   //Next line added by Steve Shedd
   learningRate = .25f;

   //Next line added by Steve Shedd
   diff = new float[NumOutputs];

   // Randomize weights here:
   randomizeWeights();
}
```

```
public void LoadTrainingCases()
{

   NumTraining = NeuralFile.NumTraining;
   int ic = 0, oc = 0;

   if ( NumTraining > 0 )
   {
      InputTraining  = new float[NumTraining * NumInputs];
      OutputTraining = new float[NumTraining * NumOutputs];
   }

   for ( int k = 0; k < NumTraining; k++ )
   {
      for ( int i = 0; i < NumInputs; i++ )
      {
         InputTraining[ic++] = NeuralFile.GetInput( k, i );
      }
      for ( int o = 0; o < NumOutputs; o++ )
      {
         OutputTraining[oc++] = NeuralFile.GetOutput( k, o );
      }

   }
}


public void Save( String output_file )
{

   NeuralFile = new NNfile();
   NeuralFile.NumHidden = this.NumHidden;
   NeuralFile.NumInput = this.NumInputs;
   NeuralFile.NumOutput = this.NumOutputs;
   NeuralFile.NumTraining = this.NumTraining;
   NeuralFile.NumLayers = 3;
   NeuralFile.NumNeuronsPerLayer = new int[3];
   NeuralFile.NumNeuronsPerLayer[0] = NeuralFile.NumInput;
   NeuralFile.NumNeuronsPerLayer[1] = NeuralFile.NumHidden;
   NeuralFile.NumNeuronsPerLayer[2] = NeuralFile.NumOutput;
   NeuralFile.SpecialFlag = 0;
   //NeuralFile.BaseIndex
   //NeuralFile.TopIndex
   //NeuralFile.WeightFlag

   NeuralFile.Save( output_file, InputTraining, OutputTraining, W1, W2 );

}


public void randomizeWeights()
{

   double x = Math.pow( NumHidden, ( 1 / NumInputs ) );
   float scale_factor = 0.7f * ( float )x;

   // Randomize weights here:
   for ( int ii = 0; ii < NumInputs; ii++ )
   {
      for ( int hh = 0; hh < NumHidden; hh++ )
      {
         //Changed by Steve Shedd
         W1[ii] [hh] = ( float )Math.random() - 0.5f;
         W1[ii] [hh] = ( scale_factor * W1[ii] [hh] ) / Math.abs( W1[ii] [hh] );
      }
   }
   for ( int hh = 0; hh < NumHidden; hh++ )
   {
```

```
      for ( int oo = 0; oo < NumOutputs; oo++ )
      {
         //Changed by Steve Shedd
         W2[hh] [oo] = ( float )Math.random() - 0.5f;
      }
   }

}


public void ForwardPass()
{

   int i, h, o;

   for ( h = 0; h < NumHidden; h++ )
   {
      //Modified by Steve Shedd - change Hidden[h] to HiddenLayer_In[h]
      HiddenLayer_In[h] = hidden_bias;
   }

   for ( i = 0; i < NumInputs; i++ )
   {
      for ( h = 0; h < NumHidden; h++ )
      {
         //Modified by Steve Shedd - change Hidden[h] to HiddenLayer_In[h]
         HiddenLayer_In[h] += Inputs[i] * W1[i] [h];
      }
   }

   //Added by Steve Shedd - perform the activiation function for each input on
   //the hidden layer nodes.
   for ( h = 0; h < NumHidden; h++ )
   {
      HiddenLayer_Out[h] = Sigmoid( HiddenLayer_In[h] );
   }

   for ( o = 0; o < NumOutputs; o++ )
   {
      //Modified by Steve Shedd - change Outputs[o] to OutputLayer_In[o]
      OutputLayer_In[o] = output_bias;
   }

   for ( h = 0; h < NumHidden; h++ )
   {
      for ( o = 0; o < NumOutputs; o++ )
      {
         //Modified by Steve Shedd - change Outputs[o] to OutputLayer_In[o]
         OutputLayer_In[o] += HiddenLayer_Out[h] * W2[h] [o];
      }
   }

   for ( o = 0; o < NumOutputs; o++ )
   {
      //Modified by Steve Shedd - change Outputs[o] to OutputLayer_Out[o]
      OutputLayer_Out[o] = Sigmoid( OutputLayer_In[o] );
   }
}


public float Train( float lr )
{

   return Train( InputTraining, OutputTraining, NumTraining, lr );

}


public float Train( float ins[], float outs[], int num_cases, float lr )
```

```
{
   //copy the ins and outs arrays to the InputTraining and Output
   //Trainig arrays.

   InputTraining = new float[ this.NumInputs * num_cases ];
   OutputTraining = new float[ this.NumOutputs * num_cases ];

   for ( int i = 0; i < ins.length; i++ )
   {
      InputTraining[i] = ins[i];
   }

   for ( int j = 0; j < outs.length; j++ )
   {
      OutputTraining[j] = outs[j];
   }


   int i, h, o;
   int in_count = 0, out_count = 0;
   float inst_error[] = new float[num_cases];

   avgNetError = 0;

   if ( lr != 0 )
   {
      learningRate = lr;
   }

   for ( int example = 0; example < num_cases; example++ )
   {

      //Set instantaneous error for this training pattern to zero.
      inst_error[example] = 0.0f;

      if ( IgnoreTraining != null )
      {
         if ( IgnoreTraining[example] )
         {
            continue;  // skip this case
         }
      }

      // zero out error arrays:
      for ( h = 0; h < NumHidden; h++ )
      { hidden_errors[h] = 0.0f; }
      for ( o = 0; o < NumOutputs; o++ )
      { output_errors[o] = 0.0f; }

      // copy the input values:
      for ( i = 0; i < NumInputs; i++ )
      {
         Inputs[i] = ins[in_count++];
      }

      /*
      if (IgnoreInput != null) {
        for (int ii = 0; ii < NumInputs; ii++) {
           if (IgnoreInput[ii]) {
             for (int hh = 0; hh < NumHidden; hh++) {
                W1[ii][hh] = 0;
             }
           }
        }
      }
      */

      // perform a forward pass through the network:
```

```
ForwardPass();

//if (MyGUI != null)  MyGUI.repaint();

//for loop below implements the delta rule for determining error in the
//output nodes.
for ( o = 0; o < NumOutputs; o++ )
{

   //Next line added by Steve Shedd - Calculate difference between desired
   //output and actual output of the output layer node.
   //change Outputs[o] to OutputLayer_Out[o]
   diff[o] = outs[out_count++] - OutputLayer_Out[o];

   //Error information term
   output_errors[o] = diff[o] * SigmoidP( OutputLayer_In[o] );

}

//Assigning error to each hidden layer node
for ( h = 0; h < NumHidden; h++ )
{
   for ( o = 0; o < NumOutputs; o++ )
   {

      //Sum of delta inputs from output layer
      hidden_errors[h] += output_errors[o] * W2[h] [o];

   }
}

//For loop below implements the delta rule for determining error in the
//hidden layers.
for ( h = 0; h < NumHidden; h++ )
{

   //Modified by Steve Shedd - changed Hidden[h] parameter of
   //SigmoidP function to HiddenLayer_In[h].

   //Error Information Term for the hidden layer
   hidden_errors[h] = hidden_errors[h] * SigmoidP( HiddenLayer_In[h] );

}

// update the hidden to output weights:
for ( o = 0; o < NumOutputs; o++ )
{
   for ( h = 0; h < NumHidden; h++ )
   {

      //Modified by Steve Shedd - Original equations looked like this:
      //
      //W2[h][o] += learningRate * output_errors[o] * Hidden[h]
      //
      //However, Hidden array is populated in ForwardPass function and
      //represents the pre-Activation function input to the hidden node,
      //not the output. In the function below, the delta rule calls for
      //the hidden nodes output. So, the new array
      //HiddenLayer_Out[] is used.

      W2[h] [o] += learningRate * output_errors[o] * HiddenLayer_Out[h];

   }
}

// update the input to hidden weights:
for ( h = 0; h < NumHidden; h++ )
{
```

```
         for ( i = 0; i < NumInputs; i++ )
         {

            //Modified by Steve Shedd - Original equations looked like this:
            //
            //W2[h][o] += learningRate * hidden_errors[h] * Inputs[i]
            //
            //However, Inputs array holds the pre-summed and pre-weighted
            //inputs to the hidden node. In otherwords, the Inputs array holds
            //the raw input pattern. In the function below, the delta rule
            //calls for the actual input to the hidden layer node.This input
            //is produced in the ForwardPass function and stored in the
            //array Hidden[].

            W1[i] [h] += learningRate * hidden_errors[h] * Inputs[i];

         }
      }

      for ( o = 0; o < NumOutputs; o++ )
      {

         //Modified by Steve Shedd - Original equations looked like this:
         //
         //error += Math.abs(output_errors[o]);
         //
         //However, equation should be the sum of the squares of the node errors.

         inst_error[example] += ( float )Math.pow( ( double )diff[o], 2 );

         //Instantaneous error = error/2
         //inst_error[example] = inst_error[example]/2;

      }


   }

   //returns average network error over entire training set
   for ( int j = 0; j < num_cases; j++ )
   {
      avgNetError += inst_error[j];
   }

   return avgNetError; // num_cases;

}


/** ****************************************************************************
 * The Sigmoid Method below is also known as the logistic function or
 * transform function
 ***************************************************************************** */
// In both Sigmoid and SigmoidP methods below, I've removed the original
// 0.5f shift factor which defined the range from -0.5f to 0.5f.
// The new range is 0.0f to 1.0f.
protected float Sigmoid( float x )
{

   return ( float )( ( 1.0f / ( 1.0f + Math.exp( ( double )( -x ) ) ) ) );//-0.5f;

}


/** ****************************************************************************
 * The SigmoidP function is simply the derivative of the Sigmoid function, or
 * logistic function.
 ***************************************************************************** */
protected float SigmoidP( float x )
```

```
   {
      double z = Sigmoid( x ); // + 0.5f;
      return ( float )( z * ( 1.0f - z ) );
   }
}
```

B. Class mwa.ai.neural.NNfile

```
//***************************************************************************
// Filename:.............NNFile.java
// Parent Project:.......SEMANTIC AND SYNTACTIC OBJECT CORRELATION IN THE OOMI IDE,
//                       Master of Computer Science Thesis Project
// Original Compiler:....Sun JDK 1.3.1_04
// Author:...............Mark Watson, in "Intelligent Java Applications"
// Date Created:.........8/5/1996
// Modified by:..........LT Steve Shedd, USN
// Company:..............Naval Postgraduate School, Monterey, California
// Date:.................September 2002
// Note:.................File modified for three reasons:
//
//         (1) To confrom to JDK 1.3.1 and deprecated methods
//         (2) To improve readability (improved commenting and layout)
//         (3) Needs of thesis work
//
// Specific comments are provided where important code has been changed from
// the original.  Deletions of minor code snippets, like System.out.println
// calls, are not commented.
//***************************************************************************

package mwa.ai.neural;

import java.io.*;
import java.net.URL;
import java.util.*;


//import java.lang.*;

class FileFormatException extends Exception
{
   public FileFormatException( String str )
   {
      super( str );
   }
}


public class NNfile
{

   public int NumLayers;

   public int NumNeuronsPerLayer[];

   public int NumInput, NumHidden, NumOutput, NumTraining;

   public int WeightFlag;

   public int SpecialFlag;

   public int BaseIndex, TopIndex;

   private float data[];

   public File inputFile = null;
```

```
public NNfile()
{

   NumLayers = NumInput = NumHidden = NumOutput = 0;
}


public NNfile( String input_file )
{

   data = new float[40000];
   TopIndex = 0;
   FileInputStream is = null;

   try
   {
      is = new FileInputStream( input_file );
      inputFile = new File( input_file );

   }
   catch ( Exception E )
   {
      System.out.println( "can not open file " + input_file );
   }

   try
   {
      if ( is != null )
      {
         ReadFile( is );
         is.close();
      }
   }
   catch ( Exception E )
   {
      System.out.println( "can not process file" );
   }

   System.out.println( "Done with ReadFile, calling ParseData..." );
   ParseData();
   OutputStream f = null;
   System.out.println( "Done with ParseData(), write output..." );

}


void ParseData()
{
   int k = 0;
   NumLayers = ( int )data[k++];
   NumNeuronsPerLayer = new int[NumLayers];

   for ( int i = 0; i < NumLayers; i++ )
   {
      NumNeuronsPerLayer[i] = ( int )data[k++];
   }

   NumInput    = NumNeuronsPerLayer[0];
   NumHidden   = NumNeuronsPerLayer[1];
   NumOutput   = NumNeuronsPerLayer[2];
   WeightFlag  = ( int )data[k++];

   if ( WeightFlag == 0 )
   {
      // Make room in data array for any weights added in later:
      int NumW = NumInput * NumHidden + NumHidden * NumOutput;
```

```
      for ( int i = TopIndex; i > 6; i-- )
      {
         data[i + NumW] = data[i];
      }
      TopIndex += NumW;
   }

   SpecialFlag = ( int )data[k++];
   NumTraining = ( int )data[k++];
   BaseIndex = k;
}


// To get weights:
public float GetW1( int input, int hidden )
{

   if ( WeightFlag == 0 )
   {
      return 0.0f;
   }

   return data[BaseIndex +
   + input * NumHidden + hidden];
}


public float GetW2( int hidden, int output )
{

   if ( WeightFlag == 0 )
   {
      return 0.0f;
   }

   return data[BaseIndex +
   + NumInput * NumHidden +
   hidden * NumOutput + output];
}


// To set weights:
public void SetW1( int input, int hidden, float x )
{

   WeightFlag = 1; // set this so save() will save weights
   data[BaseIndex +
   + input * NumHidden + hidden] = x;
}


public void SetW2( int hidden, int output, float x )
{

   WeightFlag = 1; // set this so save() will save weights
   data[BaseIndex +
   NumInput * NumHidden +
   + hidden * NumOutput + output] = x;
}


// To get any application specific data:
public float GetSpecial( int i )
{

   if ( SpecialFlag == 0 )
   {
      return 0.0f;
   }
```

```
   return data[BaseIndex +
   NumInput * NumHidden + NumHidden * NumOutput + i];
}


// To add application specific data:
public void AddSpecial( float x )
{

   // Make room in data array for a new special data value:
   int index = BaseIndex+NumInput*NumHidden+NumHidden*NumOutput+SpecialFlag;

   for ( int i = TopIndex - 1; i >= index; i-- )
   {
      data[i + 1] = data[i];
   }

   TopIndex++;
   data[index] = x;
   SpecialFlag++;
}


// To get training cases:
public float GetInput( int training_case, int neuron_index )
{

   return data[BaseIndex +
   NumInput * NumHidden + NumHidden * NumOutput +
   SpecialFlag +
   training_case * ( NumInput + NumOutput ) +
   neuron_index];
}


public float GetOutput( int training_case, int neuron_index )
{

   return data[BaseIndex +
   NumInput * NumHidden + NumHidden * NumOutput +
   SpecialFlag +
   training_case * ( NumInput + NumOutput ) +
   NumInput +
   neuron_index];
}


public void RemoveTraining( int num )
{

   if ( num < 0 || num >= NumTraining )
   {
      System.out.println( "Error in RemoveTraining(" + num + ")" );
      return;
   }

   int index = BaseIndex +
   NumInput * NumHidden + NumHidden * NumOutput +
   SpecialFlag +
   num * ( NumInput + NumOutput );

   for ( int i = index; i <= TopIndex - NumInput - NumOutput; i++ )
   {
      data[i] = data[i + NumInput + NumOutput];
   }

   TopIndex -= NumInput + NumOutput;
   NumTraining--;
```

```
}


public void AddTraining( float inputs[], float outputs[] )
{
   for ( int i = 0; i < NumInput; i++ )
   {
      data[TopIndex++] = inputs[i];
   }
   for ( int o = 0; o < NumOutput; o++ )
   {
      data[TopIndex++] = outputs[o];
   }
   NumTraining++;
}


void ReadFile( InputStream inp ) throws IOException, FileFormatException
{

   System.out.println( "Entered ReadFile" );

   Reader r = new BufferedReader( new InputStreamReader( inp ) );
   StreamTokenizer st = new StreamTokenizer( r );
   st.commentChar( '#' );
   st.eolIsSignificant( false );
   st.parseNumbers();
   System.out.println( "Before while" );

   process:
   while ( true )
   {
      switch ( st.nextToken() )
      {
         case StreamTokenizer.TT_EOL:
            System.out.println( "EOF found" );
            break process;
         case StreamTokenizer.TT_NUMBER:
            float x = ( float )st.nval;
            data[TopIndex++] = x;
            break;
         default:
            System.out.println( "Token (default):" + st.sval );
            break process;
      }
   }

   System.out.println( "Done with while loop" );
   inp.close();

   if ( st.ttype != StreamTokenizer.TT_EOF )
   { throw new FileFormatException( st.toString() ); }
}


public void Save( String save_file_name, float w1[] [], float w2[] [] )
{
   try
   {
      FileOutputStream f = new FileOutputStream( save_file_name );
      PrintStream ps = new PrintStream( f );
      ps.println( "#  Neural network data written by NNfile\n" );
      ps.println( NumLayers + "  # number of neuron layers" );
      for ( int i = 0; i < NumLayers; i++ )
      {
         ps.println( NumNeuronsPerLayer[i] + "  # neurons in layer " + i );
      }
      ps.println( "1  # weight flag" ); // always write out weights
      ps.println( SpecialFlag + "  # special data flag" );
```

```
         ps.println( NumTraining + "  # number of training cases in file" );

         ps.println( "\n# Input layer to hidden layer weights:\n" );
         for ( int i = 0; i < NumInput; i++ )
         {
            for ( int h = 0; h < NumHidden; h++ )
            {
               ps.print( w1[i] [h] + " " );
            }
            ps.print( "\n" );
         }

         ps.println( "\n# Hidden layer to output layer weights:\n" );
         for ( int h = 0; h < NumHidden; h++ )
         {
            for ( int o = 0; o < NumOutput; o++ )
            {
               ps.print( w2[h] [o] + " " );
            }
            ps.print( "\n" );
         }

         if ( SpecialFlag > 0 )
         {
            ps.println( "\n# Special network data:\n" );
            for ( int i = 0; i < SpecialFlag; i++ )
            {
               ps.println( GetSpecial( i ) + " " );
            }
            ps.println( "\n" );
         }

         ps.println( "\n# Training data:\n" );

         for ( int i = 0; i < NumTraining; i++ )
         {
            for ( int j = 0; j < NumInput; j++ )
            {
               ps.print( GetInput( i, j ) + " " );
            }
            ps.print( "   " );
            for ( int j = 0; j < NumOutput; j++ )
            {
               ps.print( GetOutput( i, j ) + " " );
            }
            ps.println( "" );
         }

         System.out.println( "Done writing to output file." );
         ps.close();
         f.close();

      }
      catch ( Exception E )
      {
         System.out.println( "can not process the file " + save_file_name );
      }
   }


   public void Save( String save_file_name, float ins[], float outs[], float w1[] [],
float w2[] [] )
   {
      try
      {
         FileOutputStream f = new FileOutputStream( save_file_name );
         PrintStream ps = new PrintStream( f );
         ps.println( "#  Neural network data written by NNfile\n" );
         ps.println( NumLayers + "  # number of neuron layers" );
```

```
            for ( int i = 0; i < NumLayers; i++ )
            {
               ps.println( NumNeuronsPerLayer[i] + "  # neurons in layer " + i );
            }
            ps.println( "1  # weight flag" ); // always write out weights
            ps.println( SpecialFlag + "  # special data flag" );
            ps.println( NumTraining + "  # number of training cases in file" );

            ps.println( "\n# Input layer to hidden layer weights:\n" );
            for ( int i = 0; i < NumInput; i++ )
            {
               for ( int h = 0; h < NumHidden; h++ )
               {
                  ps.print( w1[i] [h] + " " );
               }
               ps.print( "\n" );
            }

            ps.println( "\n# Hidden layer to output layer weights:\n" );
            for ( int h = 0; h < NumHidden; h++ )
            {
               for ( int o = 0; o < NumOutput; o++ )
               {
                  ps.print( w2[h] [o] + " " );
               }
               ps.print( "\n" );
            }

            if ( SpecialFlag > 0 )
            {
               ps.println( "\n# Special network data:\n" );
               for ( int i = 0; i < SpecialFlag; i++ )
               {
                  ps.println( GetSpecial( i ) + " " );
               }
               ps.println( "\n" );
            }

            ps.println( "\n# Training data:\n" );

            int in_count = 0;
            int out_count = 0;

            for ( int i = 0; i < NumTraining; i++ )
            {
               for ( int j = 0; j < NumInput; j++ )
               {
                  ps.print( ins[in_count++] + " " );
               }
               ps.print( "   " );
               for ( int j = 0; j < NumOutput; j++ )
               {
                  ps.print( outs[out_count++] + " " );
               }
               ps.println( "" );
            }

            System.out.println( "Done writing to output file." );
            ps.close();
            f.close();

         }
         catch ( Exception E )
         {
            System.out.println( "can not process the file " + save_file_name );
         }
      }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. TEST XML SCHEMAS

A. COMPONENT CLASS REPRESENTATION (CCR) XML SCHEMAS

1. Armored Fighting Vehicle

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/systemD"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:systemD="http://nps.navy.mil/cs/oomi/systemD" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xsd:element name="armoredFightingVehicle">
		<xsd:complexType>
			<xsd:annotation>
				<xsd:documentation> This sample schema provided for example
in Figure II-1.  The armoredFightingVehicle of System D models the ground combat vehicle
real-world entity.</xsd:documentation>
			</xsd:annotation>
			<xsd:sequence>
				<xsd:element name="afvClassification"
type="systemD:afvClassificationType"/>
				<xsd:element ref="systemD:afvLocation"/>
				<xsd:element ref="systemD:afvObsTime"/>
				<xsd:element name="afvStatus" type="systemD:afvStatusType"
minOccurs="0"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
	<xsd:simpleType name="afvClassificationType">
		<xsd:restriction base="xsd:string">
			<xsd:minLength value="5"/>
			<xsd:maxLength value="14"/>
			<xsd:enumeration value="battleTank"/>
			<xsd:enumeration value="rocketLauncher"/>
			<xsd:enumeration value="truck"/>
			<xsd:enumeration value="unknown"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:element name="afvLocation">
		<xsd:complexType>
			<xsd:annotation>
				<xsd:documentation/>
			</xsd:annotation>
			<xsd:sequence>
				<xsd:element name="latitude" type="systemD:latitudeType"/>
				<xsd:element name="longitude"
type="systemD:longitudeType"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="afvObsTime">
		<xsd:complexType>
			<xsd:annotation>
				<xsd:documentation>The day of a month and timekeeping in
hours and minutes of a calendar day, using the 24-hour clock system referenced to
Greenwich Mean Time (GMT).</xsd:documentation>
			</xsd:annotation>
			<xsd:sequence>
				<xsd:element name="day" type="systemD:dayType"/>
				<xsd:element name="hourTime" type="systemD:hourTimeType"/>
				<xsd:element name="minuteTime"
type="systemD:minuteTimeType"/>
				<xsd:element name="stdTimeZone"
type="systemD:stdTimeZoneType"/>
			</xsd:sequence>
		</xsd:complexType>
```

```
	</xsd:element>
	<xsd:simpleType name="afvStatusType">
		<xsd:restriction base="xsd:string">
			<xsd:minLength value="7"/>
			<xsd:maxLength value="11"/>
			<xsd:enumeration value="operational"/>
			<xsd:enumeration value="damaged"/>
			<xsd:enumeration value="destroyed"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="dayType">
		<xsd:restriction base="xsd:string">
			<xsd:length value="2"/>
			<xsd:pattern value="[0-3]{1}[0-9]{1}"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="hourTimeType">
		<xsd:annotation>
			<xsd:documentation/>
		</xsd:annotation>
		<xsd:restriction base="xsd:string">
			<xsd:length value="2"/>
			<xsd:pattern value="[0-2]{1}[0-9]{1}"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="minuteTimeType">
		<xsd:annotation>
			<xsd:documentation/>
		</xsd:annotation>
		<xsd:restriction base="xsd:string">
			<xsd:length value="2"/>
			<xsd:pattern value="[0-5]{1}[0-9]{1}"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="stdTimeZoneType">
		<xsd:annotation>
			<xsd:documentation/>
		</xsd:annotation>
		<xsd:restriction base="xsd:string">
			<xsd:length value="1"/>
			<xsd:pattern value="[Z]{1}"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:complexType name="latitudeType">
		<xsd:annotation>
			<xsd:documentation/>
		</xsd:annotation>
		<xsd:sequence>
			<xsd:element name="latDegMinSec" type="systemD:latDegMinSecType"/>
			<xsd:element name="nsHemisphere" type="systemD:nsHemisphereType"/>
		</xsd:sequence>
	</xsd:complexType>
	<xsd:complexType name="longitudeType">
		<xsd:annotation>
			<xsd:documentation/>
		</xsd:annotation>
		<xsd:sequence>
			<xsd:element name="longDegMinSec"
type="systemD:longDegMinSecType"/>
			<xsd:element name="ewHemisphere" type="systemD:ewHemisphereType"/>
		</xsd:sequence>
	</xsd:complexType>
	<xsd:simpleType name="latDegMinSecType">
		<xsd:restriction base="xsd:string">
			<xsd:length value="6"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="longDegMinSecType">
		<xsd:restriction base="xsd:string">
```

```
                        <xsd:length value="7"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="nsHemisphereType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:enumeration value="N"/>
                        <xsd:enumeration value="S"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="ewHemisphereType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:enumeration value="E"/>
                        <xsd:enumeration value="W"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

2. Mechanized Combat Vehicle

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/systemB"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:systemB="http://nps.navy.mil/cs/oomi/systemB" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <xsd:element name="mechanizedCombatVehicle">
                <xsd:annotation>
                        <xsd:documentation> This sample schema provided for example in
Figure II-1.  The mechanizedCombatVehicle of System B models the ground combat vehicle
real-world entity.</xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation>A mechanizedCombatVehicle msgtype
provides System B model of ground combat vehicle real-world entity for the example in
Figure III-3.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="mcvType" type="systemB:mcvType"/>
                                <xsd:element ref="systemB:mcvLocation"/>
                                <xsd:element ref="systemB:mcvTime"/>
                                <xsd:element name="mcvRadius"
type="systemB:distanceInKmType" minOccurs="0"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="mcvType">
                <xsd:restriction base="xsd:string">
                        <xsd:minLength value="4"/>
                        <xsd:maxLength value="16"/>
                        <xsd:enumeration value="tank"/>
                        <xsd:enumeration value="personnelCarrier"/>
                        <xsd:enumeration value="reconVehicle"/>
                        <xsd:enumeration value="unknown"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:element name="mcvLocation">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation/>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="utmZone" type="systemB:utmZoneType"/>
                                <xsd:element name="mgrsEasting"
type="systemB:mgrsEastingType"/>
                                <xsd:element name="mgrsNorthing"
type="systemB:mgrsNorthingType"/>
```

```
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="mcvTime">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation>The day of a month and timekeeping in
hours and minutes of a calendar day, using the 24-hour clock system and an associated
time zone.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="day" type="systemB:dayType"/>
                                <xsd:element name="hourTime" type="systemB:hourTimeType"/>
                                <xsd:element name="minuteTime"
type="systemB:minuteTimeType"/>
                                <xsd:element name="localTimeZone"
type="systemB:localTimeZoneType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:complexType name="utmZoneType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element name="eastWest" type="systemB:eastWestZoneType"/>
                        <xsd:element name="northSouth" type="systemB:northSouthZoneType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:simpleType name="eastWestZoneType">
                <xsd:restriction base="xsd:integer">
                        <xsd:minInclusive value="1"/>
                        <xsd:maxInclusive value="60"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="northSouthZoneType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:pattern value="[A-V]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="mgrsEastingType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="4"/>
                        <xsd:pattern value="[A-Z]{1}[0-9]{3}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="mgrsNorthingType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="4"/>
                        <xsd:pattern value="[A-V]{1}[0-9]{3}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="dayType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-3]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="hourTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-2]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="minuteTimeType">
```

```
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-5]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="localTimeZoneType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:pattern value="[A-Z]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="distanceInKmType">
                <xsd:restriction base="xsd:integer">
                        <xsd:minInclusive value="0"/>
                        <xsd:maxInclusive value="1200"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

B. FEDERATION CLASS REPRESENTATION (FCR) XML

1. Ground Combat Vehicle View 1

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/fiomA"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:fiomA="http://nps.navy.mil/cs/oomi/fiomA" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <xsd:element name="groundCombatVehicleView1">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation>This sample schema provided for example
in Figure II-1.  groundCombatVehicleView1 provides same perspective of ground combat
vehicle real-world entity as System B mechanizedCombatVehicle model.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="vehicle" type="fiomA:vehicleType"
minOccurs="0"/>
                                <xsd:element ref="fiomA:position" minOccurs="0"/>
                                <xsd:element ref="fiomA:time" minOccurs="0"/>
                                <xsd:element name="range" type="fiomA:distanceInNmType"
minOccurs="0"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="vehicleType">
                <xsd:restriction base="xsd:string">
                        <xsd:minLength value="5"/>
                        <xsd:maxLength value="16"/>
                        <xsd:enumeration value="battleTank"/>
                        <xsd:enumeration value="rocketLauncher"/>
                        <xsd:enumeration value="truck"/>
                        <xsd:enumeration value="personnelCarrier"/>
                        <xsd:enumeration value="reconVehicle"/>
                        <xsd:enumeration value="unknown"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:element name="position">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation/>
                        </xsd:annotation>
```

```
                        <xsd:sequence>
                                <xsd:element name="latitude" type="fiomA:latitudeType"/>
                                <xsd:element name="longitude" type="fiomA:longitudeType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="time">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation>The day of a month and timekeeping in
hours and minutes of a calendar day, using the 24-hour clock system and an associated
time zone.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="stdDay" type="fiomA:stdDayType"/>
                                <xsd:element name="stdHourTime"
type="fiomA:stdHourTimeType"/>
                                <xsd:element name="stdMinuteTime"
type="fiomA:stdMinuteTimeType"/>
                                <xsd:element name="stdTimeZone"
type="fiomA:stdTimeZoneType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="distanceInNmType">
                <xsd:restriction base="xsd:integer">
                        <xsd:minInclusive value="0"/>
                        <xsd:maxInclusive value="1000"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdDayType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-3]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdHourTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-2]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdMinuteTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-5]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdTimeZoneType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:pattern value="[Z]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="latitudeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element name="latDegMinSec" type="fiomA:latDegMinSecType"/>
```

```
                        <xsd:element name="nsHemisphere" type="fiomA:nsHemisphereType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="longitudeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element name="longDegMinSec" type="fiomA:longDegMinSecType"/>
                        <xsd:element name="ewHemisphere" type="fiomA:ewHemisphereType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:simpleType name="latDegMinSecType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="6"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="longDegMinSecType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="7"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="nsHemisphereType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:enumeration value="N"/>
                        <xsd:enumeration value="S"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="ewHemisphereType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:enumeration value="E"/>
                        <xsd:enumeration value="W"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

2. Ground Combat Vehicle View 2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/fiomA"
xmlns:fiomA="http://nps.navy.mil/cs/oomi/fiomA"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <xsd:element name="groundCombatVehicleView2">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation> This sample schema provided for example
in Figure II-1.  groundCombatVehicleView2 provides same perspective of System C
armoredMilitaryVehicle model of ground combat vehicle real-world entity for the example
in Figure II-1.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="vehicle" type="fiomA:vehicleType"
minOccurs="0"/>
                                <xsd:element ref="fiomA:position" minOccurs="0"/>
                                <xsd:element ref="fiomA:time" minOccurs="0"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="vehicleType">
                <xsd:restriction base="xsd:string">
                        <xsd:minLength value="5"/>
                        <xsd:maxLength value="16"/>
                        <xsd:enumeration value="battleTank"/>
                        <xsd:enumeration value="rocketLauncher"/>
                        <xsd:enumeration value="truck"/>
```

```
                        <xsd:enumeration value="personnelCarrier"/>
                        <xsd:enumeration value="reconVehicle"/>
                        <xsd:enumeration value="unknown"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:element name="position">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation/>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="latitude" type="fiomA:latitudeType"/>
                                <xsd:element name="longitude" type="fiomA:longitudeType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="time">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation>The day of a month and timekeeping in
hours and minutes of a calendar day, using the 24-hour clock system and an associated
time zone.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="stdDay" type="fiomA:stdDayType"/>
                                <xsd:element name="stdHourTime"
type="fiomA:stdHourTimeType"/>
                                <xsd:element name="stdMinuteTime"
type="fiomA:stdMinuteTimeType"/>
                                <xsd:element name="stdTimeZone"
type="fiomA:stdTimeZoneType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="stdDayType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-3]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdHourTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-2]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdMinuteTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-5]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdTimeZoneType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:pattern value="[Z]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="latitudeType">
                <xsd:annotation>
```

```
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element name="latDegMinSec" type="fiomA:latDegMinSecType"/>
                        <xsd:element name="nsHemisphere" type="fiomA:nsHemisphereType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="longitudeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:sequence>
                        <xsd:element name="longDegMinSec" type="fiomA:longDegMinSecType"/>
                        <xsd:element name="ewHemisphere" type="fiomA:ewHemisphereType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:simpleType name="latDegMinSecType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="6"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="longDegMinSecType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="7"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="nsHemisphereType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:enumeration value="N"/>
                        <xsd:enumeration value="S"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="ewHemisphereType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
                        <xsd:enumeration value="E"/>
                        <xsd:enumeration value="W"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

3. Ground Combat Vehicle View 3

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/fiomA"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:fiomA="http://nps.navy.mil/cs/oomi/fiomA" elementFormDefault="qualified"
attributeFormDefault="unqualified">
        <xsd:element name="groundCombatVehicleView3">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation> This sample schema provided for example
in Figure II-1.  groundCombatVehicleView3 provides same perspective of ground combat
vehicle real-world entity as System D armoredFightingVehicle model.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="vehicle" type="fiomA:vehicleType"
minOccurs="0"/>
                                <xsd:element ref="fiomA:position"/>
                                <xsd:element ref="fiomA:time"/>
                                <xsd:element name="status" type="fiomA:statusType"
minOccurs="0"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="vehicleType">
                <xsd:restriction base="xsd:string">
```

```
                        <xsd:minLength value="5"/>
                        <xsd:maxLength value="16"/>
                        <xsd:enumeration value="battleTank"/>
                        <xsd:enumeration value="rocketLauncher"/>
                        <xsd:enumeration value="truck"/>
                        <xsd:enumeration value="personnelCarrier"/>
                        <xsd:enumeration value="reconVehicle"/>
                        <xsd:enumeration value="unknown"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:element name="position">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation/>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="latitude" type="fiomA:latitudeType"/>
                                <xsd:element name="longitude" type="fiomA:longitudeType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="time">
                <xsd:complexType>
                        <xsd:annotation>
                                <xsd:documentation>The day of a month and timekeeping in
hours and minutes of a calendar day, using the 24-hour clock system and an associated
time zone.</xsd:documentation>
                        </xsd:annotation>
                        <xsd:sequence>
                                <xsd:element name="stdDay" type="fiomA:stdDayType"/>
                                <xsd:element name="stdHourTime"
type="fiomA:stdHourTimeType"/>
                                <xsd:element name="stdMinuteTime"
type="fiomA:stdMinuteTimeType"/>
                                <xsd:element name="stdTimeZone"
type="fiomA:stdTimeZoneType"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:simpleType name="stdDayType">
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-3]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdHourTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-2]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdMinuteTimeType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="2"/>
                        <xsd:pattern value="[0-5]{1}[0-9]{1}"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stdTimeZoneType">
                <xsd:annotation>
                        <xsd:documentation/>
                </xsd:annotation>
                <xsd:restriction base="xsd:string">
                        <xsd:length value="1"/>
```

```
                <xsd:pattern value="[Z]{1}"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="latitudeType">
            <xsd:annotation>
                <xsd:documentation/>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="latDegMinSec" type="fiomA:latDegMinSecType"/>
                <xsd:element name="nsHemisphere" type="fiomA:nsHemisphereType"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="longitudeType">
            <xsd:annotation>
                <xsd:documentation/>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="longDegMinSec" type="fiomA:longDegMinSecType"/>
                <xsd:element name="ewHemisphere" type="fiomA:ewHemisphereType"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:simpleType name="latDegMinSecType">
            <xsd:restriction base="xsd:string">
                <xsd:length value="6"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="longDegMinSecType">
            <xsd:restriction base="xsd:string">
                <xsd:length value="7"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="nsHemisphereType">
            <xsd:restriction base="xsd:string">
                <xsd:length value="1"/>
                <xsd:enumeration value="N"/>
                <xsd:enumeration value="S"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="ewHemisphereType">
            <xsd:restriction base="xsd:string">
                <xsd:length value="1"/>
                <xsd:enumeration value="E"/>
                <xsd:enumeration value="W"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="statusType">
            <xsd:restriction base="xsd:string">
                <xsd:minLength value="7"/>
                <xsd:maxLength value="11"/>
                <xsd:length value="3"/>
                <xsd:enumeration value="operational"/>
                <xsd:enumeration value="damaged"/>
                <xsd:enumeration value="destroyed"/>
            </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Man-Tak Shing
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

4. CAPT Paul Young
   Department of Computer Science
   United States Naval Academy
   Annapolis, Maryland

5. LT Steve Shedd
   Class 172
   Surface Warfare Officer School Command
   Newport, Rhode Island

6. Professor Luqi
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

7. LCDR Chris Eagle
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California